University of Anbar
College of Computer Science
and Information Technology
Computer Network Systems
Department

Data Structures

Lecture Six

Second Stage

First Course - 2024-2025

Muhmmad shihab muayad Shihab

Master of Computer Engineering

muhmmad.shihab@uoanbar.edu.iq

Queue

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Queue Representation

As the understand now that in queue, the access are both ends for different reasons. The following diagram given below tries to explain queue representation as data structure:

As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures. For the sake of simplicity, we shall implement queues using one-dimensional array.

Basic features of Queue

1. Like stack, queue is also an ordered list of elements of similar data types.
2. Queue is a FIFO(First in First Out) structure.
3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.
4. peek( ) function is oftenly used to return the value of first element without dequeuing it.

Basic Operations

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues:

- **enqueue**() − add (store) an item to the queue.
- **dequeue**() − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are:

- peek() − Gets the element at the front of the queue without removing it.
- isfull() − Checks if the queue is full.
- isempty() − Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by front pointer and while enqueing (or storing) data in the queue we take help of rear pointer.

Let's first learn about supportive functions of a queue:

- **peek()**

This function helps to see the data at the front of the queue. Implementation of peek() function in C programming language:

```
int peek() {
   return queue[front];
}
```

- **isfull()**

As we are using single dimension array to implement queue, we just check for the rear pointer to reach at MAXSIZE to determine that the queue is full. In case we maintain the queue in a circular linked-list, the algorithm will differ. Implementation of isfull() function in C programming language:

```
bool isfull() {
   if(rear == MAXSIZE - 1)
      return true;
   else
      return false;
}
```

- **isempty()**

If the value of front is less than MIN or 0, it tells that the queue is not yet initialized, hence empty.

```
bool isempty() {
  if(front < 0 || front > rear)
    return true;
  else
    return false;
}
```

Enqueue Operation

Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks. The following steps should be taken to enqueue (insert) data into a queue:

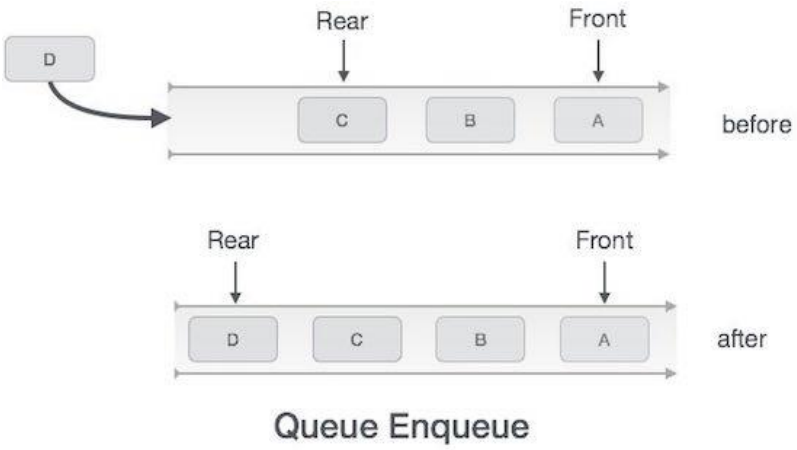
Step 1: Check if the queue is full.

Step 2: If the queue is full, produce overflow error and exit.

Step 3: If the queue is not full, increment rear pointer to point the next empty space.

Step 4: Add data element to the queue location, where the rear is pointing.

Step 5: return success.



Queue Enqueue

Sometimes, we also check to see if a queue is initialized or not, to handle any unforeseen situations. Implementation of enqueue() in C programming language:

```
int enqueue(int data)
  if(isfull())
    return 0;
  rear = rear + 1;
  queue[rear] = data;
  return 1;
```

end procedure

Dequeue Operation

Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation −

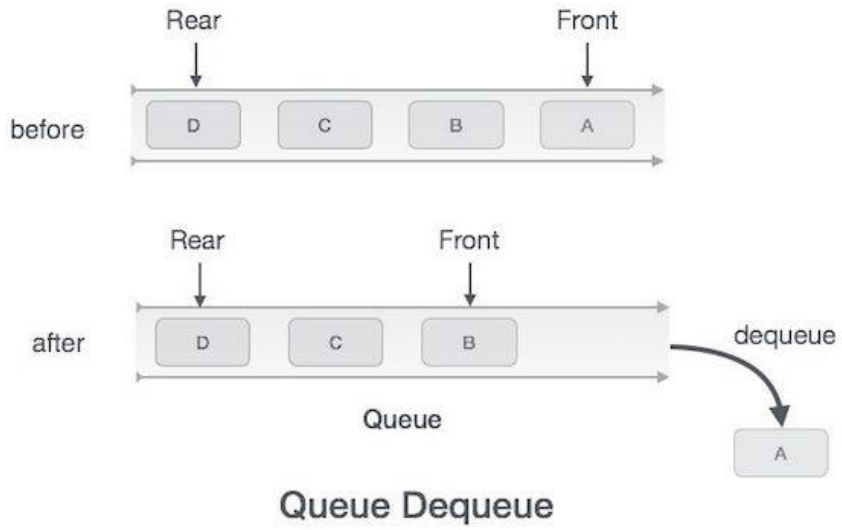
Step 1: Check if the queue is empty.

Step 2: If the queue is empty, produce underflow error and exit.

Step 3: If the queue is not empty, access the data where front is pointing.

Step 4: Increment front pointer to point to the next available data element.

Step 5: Return success.



Queue Dequeue

Implementation of dequeue() in C programming language:

```
int dequeue() {
   if(isempty())
      return 0;
   int data = queue[front];
   front = front + 1;
   return data;
}
```

Full program for Queue

```
#include<iostream.h>
const int size=5;
void enq(char q[size],int&f,int&r,char x);
```

```
void deq(char q[size],int&f,int&r,char &x);
int full_q(int r);
int empty_q(int f);
main()
{
char q[size],x;
int f=-1,r=-1,no;
do{
cout<<"\n\n 1.en_Q.\n";
cout<<" 2.de_Q.\n";
cout<<" 3.exit\n\n";
cout<<"ENTER YOUR CHOICE:";
                    cin>>no;
        switch(no)
        {
        case 1:cout<<"enter char which you want to add in Q:";
                        cin>>x;
                        enq(q,f,r,x);
                            break;
    case 2: deq(q,f,r,x);
                break;
        case 3:  break;
        }
}while(no!=3);
}

int full_q(int r)

 {
 if(r>=size-1)return 1;
        else
          return 0;
 }
int empty_q(int f)
{
  if(f==-1)return 1;
        else
         return 0;
}
```

```
void enq(char q[size],int&f,int&r,char x)
{
if(full_q(r))cout<<"Queue is full!!";
else
{
      r++;
         q[r]=x;
         if(f=-1)f=0;
         cout<<"Q["<<r<<"]="<<q[r]; }}
void deq(char q[size],int&f,int&r,char &x)
{
 if(empty_q(r))cout<<"Queue is empty!!";
 else
 {
  x=q[f];
  cout<<"Q["<<f<<"]="<<x;الطابور
  if(f==r)f=r=-1;
         else
               f++;
  }
}
```

Queue's Applications

Given that queues occur frequently in real life, it is not surprising that they are also frequently used in simulations. To model customers using a bank, for example, you could use a queue of waiting patrons. A simulation might want to ask questions such as the how the average waiting time would change if you added or removed a teller position. Queues are also used in any time collection where time of insertion is important. We have noted, for example, that a printer might want to keep the collection of pending jobs in a queue, so that they will be printed in the same order that they were submitted.

Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following examples :

1) Serving requests on a single shared resource, like a printer, CPU task scheduling etc.

2) In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
3) Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

Our software queues have counterparts in real world queues. We wait in queues to buy pizza, to enter movie theaters, to drive on a turnpike, and to ride on a roller coaster. Another important application of the queue data structure is to help us simulate and analyze such real world queues.

Many examples in a computer's operating system. Requests for services come in unpredictable order and timing, sometimes faster than they can be serviced (like at a bank window, cafeteria, etc.)

- print a file
- need a file from the disk system
- send an email

Implementation of queue

A. Implementation of a queue is also an important application of data structure. Nowadays computer handles multiuser, multiprogramming environment and time-sharing environment. In this environment a system(computer) handles several jobs at a time, to handle these jobs the concept of a queue is used.
B. To implement printer spooler so that jobs can be printed in the order of their arrival.
C. Round robin scheduling technique is implemented using queue
D. All types of customer service(like railway reservation) centers are designed using the concept of queues.

References:

- Frank Carrano, D.J. Henry: Data Abstraction and Solving with C++, 2012, 6th edition, Pearson Education, Inc.
- Mark Allen Weiss: Data Structures and Algorithm Analysis in C++, 2014, 4th edition, Pearson Education, Inc.