Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Department of Information Technology

Chapter-3
Queue

Introduction to Queue, Queue as ADT, Operations on Queue, Linear representation of queue, Circular Queue, Priority Queue, De-queue, Application of Queues.

UNIT - 3

QUEUES

A queue is an important data structure which is extensively used in computer application

Let us explain the concept of queues using the analogies given below.

- People moving on an escalator. The people who got on the escalator first will be the first one to step out of it.
- People waiting for a bus. The first person standing in the line will be the first one to get into bus
- People standing outside the ticketing window of a cinema hall. The first person in the line will get the ticket first and thus will be the first one to move out of it.
- Luggage kept on conveyor belts. The bag which was placed first will be the first to come out at the other end.
- Cars lined at a toll bridge. The first car to reach the bridge will be the first to leave.

In all these examples, we see that the element at the first position is served first. Same is the case with queue data structure. A queue is a FIFO (first-In, First-Out) data

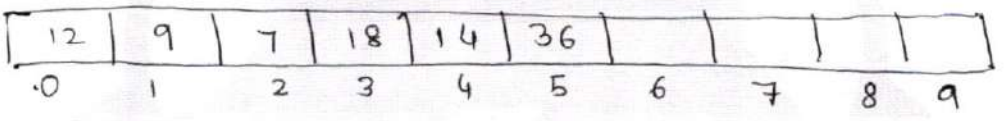
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

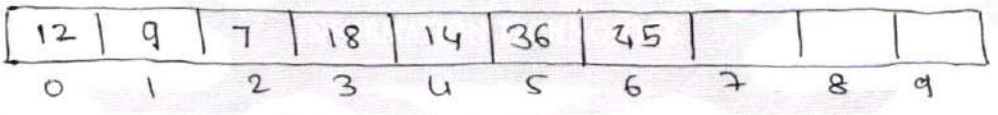
structure in which the element that is insert-ed first is the first one to be taken out. The elements in a queue are added at one end called the REAR and removed from the other end called the FRONT.

ARRAY REPRESENTATION OF QUEUES

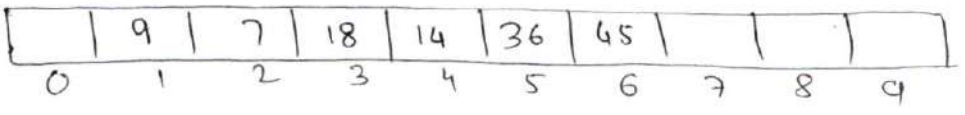
Queue can be easily represented using linear arrays. As stated earlier, every queue has FRONT and REAR variables that point to the position from where deletions and insertions can be done, respectively. The array representation of a queue is shown in fig



Queue



Queue after insertion of a new element



Queue after deletion of an element

Operation on Queues

In fig above FRONT = 0 and REAR = 5 Suppose we want to add another element with value 45, then REAR would be incremented by 1 and the value

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

would be stored at the position pointed by REAR.

The queue after addition would be as shown in above fig. Here FRONT = 0 and REAR = 6. Every time a new element has to be added, we repeat the same procedure.

If we want to delete an element from the queue, then the value of FRONT will be incremented. Deletions are done from only this end of queue. The queue after deletion will be as shown in above figure.

However, before inserting an element in a queue, we must check for OVERFLOW condition, An Overflow will occur when we try to insert an element into a queue that is already full. When REAR = MAX - 1 where MAX is the size of the queue, we have an overflow condition. Note that we have written MAX -1 because the index starts from 0.

```
Step 1 : IF REAR = MAX - 1
           Write OVERFLOW
           Goto step 4
         [ END OF IF ]
Step 2 : IF FRONT = -1 and REAR = -1
           SET FRONT = REAR = 0
         ELSE
           SET REAR = REAR + 1
         [ END OF IF ]
Step 3 : SET QUEUE [REAR] = NUM
step 4 : EXIT
```

Similarly, before deleting an element from a queue, we must check for underflow conditions. An underflow condition occurs when we try to delete an element from a queue that is already empty. IF FRONT=-1 and REAR = -1, it means there is no element in the queue.

```
Step 1 : IF FRONT = -1 OR FRONT > REAR
            WRITE UNDERFLOW
         ELSE
            SET VAL = QUEUE [FRONT]
            SET FRONT = FRONT + 1
         [END OF IF]
Step 2 : EXIT
```

Algorithm to delete an element from a queue

```
1. /*
2. * C Program to Implement a Queue using an Array
3. */
4.#include <stdio.h>
5.
6.#define MAX 50
7.int queue_array[MAX];
8.int rear = - 1;
9.int front = - 1;
10.main()
11.{
12.   int choice;
13.   while (1)
14.   {
15.      printf("1.Insert element to queue \n");
16.      printf("2.Delete element from queue \n");
17.      printf("3.Display all elements of queue \n");
18.      printf("4.Quit \n");
19.      printf("Enter your choice : ");
20.      scanf("%d", &choice);
21.      switch (choice)
22.      {
23.         case 1:
24.         enqueue();
25.         break;
26.         case 2:
27.         dqueue();
28.         break;
29.         case 3:
1
30.         display();
31.         break;
32.         case 4:
33.         exit(1);
34.         default:
35.         printf("Wrong choice \n");
36.      } /*End of switch*/
37.   } /*End of while*/
38.} /*End of main()*/
39.enqueue()
40.{
41.   int add_item;
42.   if (rear == MAX - 1)
43.   printf("Queue Overflow \n");
44.   else
45.   {
46.      if (front == - 1)
47.      /*If queue is initially empty */
```

```
48.      front = 0;
49.      printf("Inset the element in queue : ");
50.      scanf("%d", &add_item);
51.      rear = rear + 1;
52.      queue_array[rear] = add_item;
53.   }/*End of insert()*/
55.
56.dequeue()
57.{
58.   if (front == - 1 || front > rear)
59.   {
60.      printf("Queue Underflow \n");
61.      return ;
62.   }
63.   else
64.   {
65.      printf("Element deleted from queue is : %d\n",
       queue_array[front]);
66.      front = front + 1;
67.   }
68.} /*End of delete() */
69.display()
70.{
71.   int i;
72.   if (front == - 1)
73.      printf("Queue is empty \n");
2
74.   else
75.   {
76.      printf("Queue is : \n");
77.      for (i = front; i <= rear; i++)
78.         printf("%d ", queue_array[i]);
79.      printf("\n");
80.   }
81.} /*End of display() */
void peek()
{
int data;
data=queue_array[front];
printf("The peek of queue is %d",data);
}
/*
$cc pgm.c
$ a.out
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
```

```
Enter your choice : 1
Inset the element in queue : 10
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 15
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 20
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
3
Enter your choice : 1
Inset the element in queue : 30
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2
Element deleted from queue is : 10
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3
Queue is :
15 20 30
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 4*/
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

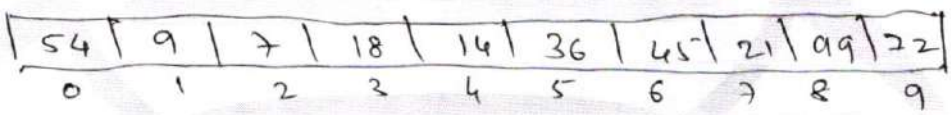
TYPES OF QUEUES

A queue data structure can be classified into the following types :-

1. Circular Queue
2. Deque
3. Priority Queue

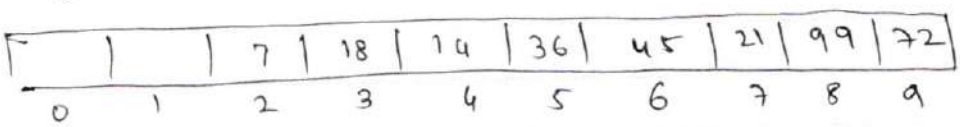
1) Circular Queues

In linear queues, we have discussed so far that insertions can be done only at one end called the REAR and deletions are always done from the other end called the FRONT. Look at the queue shown in this fig -



Linear Queue

Now, if you want to insert another value, it will not be possible because the queque is compeletely full. There is no empty space where the value can be inserted Consider a scenario in which two successive deletions are made. The queue will then be given as shown in fig



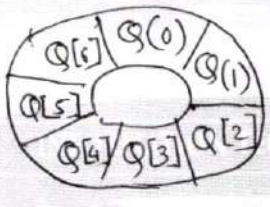
Queue after two successive deletions

Here FRONT = 2 & REAR = 9

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Suppose we want to insert a new element in the queue shown above fig. Even though there is space available, the overflow condition still exists because the condition REAR = MAX - 1 still holds true.

To resolve this problem, we have two solution First, shift the elements to the left so that the vacant space can be occupied and utilized efficiently. But this can be very time-consuming especially when the queue is quite large. The second option is to use a circular queue. In the circular queue the first index comes right after the last index. Conceptually, you can think of circular queue as shown below



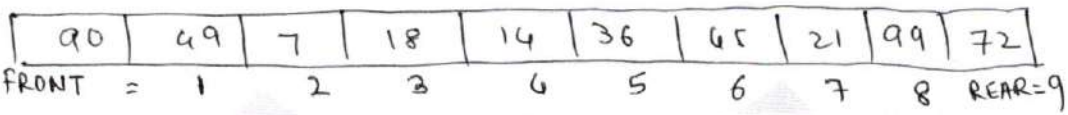
The circular queue will be full only when FRONT = 0 and REAR = MAX - 1. A circular queue is implemented in the same manner as a linear queue is implemented in the same manner as a linear queue is implemented. The only difference will be in the code that performs insertion and deletion operations. For insertion, we now have to check for the following three conditions:

- If FRONT = 0 and REAR = MAX - 1 then the circular queue is full. Look

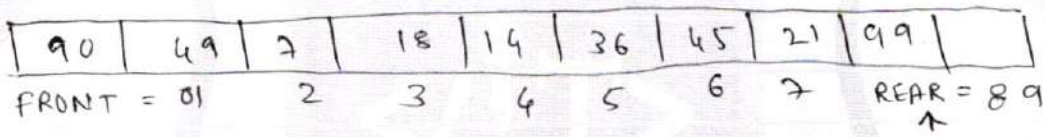
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

at the queue given below which illustrate this point

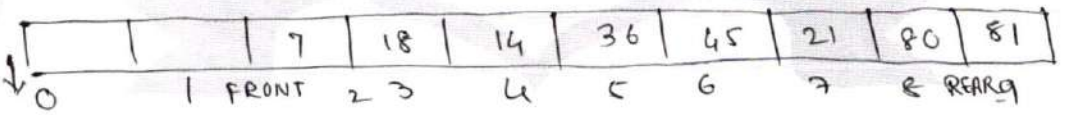


⊙ If REAR = ! = MAX - 1, then REAR will be incremented and the value will be inserted as illustrated in following fig



Increment rear so that it point to location 9 and insert the value here

⊙ If FRONT ! = 0 and REAR = MAX - 1, then it means that the queue is not full, So set REAR = 0 and insert the new element there, as shown in this following fig



Set REAR = 0 and insert the value here.

```
step 1 : IF FRONT = 0 and Rear = MAX - 1
           Write "OVERFLOW"
           Goto step 4
         [END OF IF]
Step 2 : IF FRONT = -1 and REAR = -1
           SET FRONT = REAR = 0
         ELSE IF REAR = MAX - 1 and FRONT ! = 0
           SET REAR = 0
         ELSE
           SET REAR = REAR + 1
Step 3 : SET QUEUE [ REAR] = VAL
step 4 : EXIT
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)


```
Step 1 : IF FRONT = -1
         WRITE " UNDERFLOW "
         Goto Step 4
        [ END OF IF ]
Step 2 : SET VAL = QUEUE [ FRONT ]
Step 3 : IF FRONT = REAR
            SET FRONT = REAR = -1
         ELSE
           IF FRONT = MAX - 1
           SET FRONT = 0
         ELSE
            SET FRONT = FRONT + 1
         [ END OF IF ]
       [ END OF IF ]
Step 4 : EXIT
```

```
/*
Aim:- Implementation of Circular Queue menu driven program.*/
#include<stdio.h>
#define SIZE 5

void enQueue(int);
void deQueue();
void display();

int cQueue[SIZE], front = -1, rear = -1;

int main()
{
  int choice, value;
  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("\nEnter the value to be inserted:  ");
                scanf("%d",&value);
                enQueue(value);
                break;
        case 2: deQueue();
                break;
        case 3: display();
                break;
        case 4: break;
        default: printf("\nPlease select the correct choice!!!\n");
    }
  }
}
void enQueue(int value)
{
  if((front == 0 && rear == SIZE - 1) || (front == rear+1))
    printf("\nCircular Queue is Full! Insertion not possible!!!\n");
  else{
    if(rear == SIZE-1 && front != 0)
        rear = -1;

    cQueue[++rear] = value;
    printf("\nInsertion Success!!!\n");
    if(front == -1)
        front = 0;
  }
}
```

```
void deQueue()
{
  if(front == -1 && rear == -1)
    printf("\nCircular Queue is Empty! Deletion is not possible!!!\n");
  else{
    printf("\nDeleted element : %d\n",cQueue[front++]);
    if(front == SIZE)
        front = 0;
    if(front-1 == rear)
        front = rear = -1;
  }
}
void display()
{
  if(front == -1)
    printf("\nCircular Queue is Empty!!!\n");
  else{
    int i = front;
    printf("\nCircular Queue Elements are : \n");
    if(front <= rear){
        while(i <= rear)
          printf("%d\t",cQueue[i++]);
    }
    else{
        while(i <= SIZE - 1)
          printf("%d\t", cQueue[i++]);
        i = 0;
        while(i <= rear)
          printf("%d\t",cQueue[i++]);
    }
  }
}
/*
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1

Enter the value to be inserted:  60

Circular Queue is Full! Insertion not possible!!!

****** MENU ******
1. Insert
2. Delete
3. Display
```

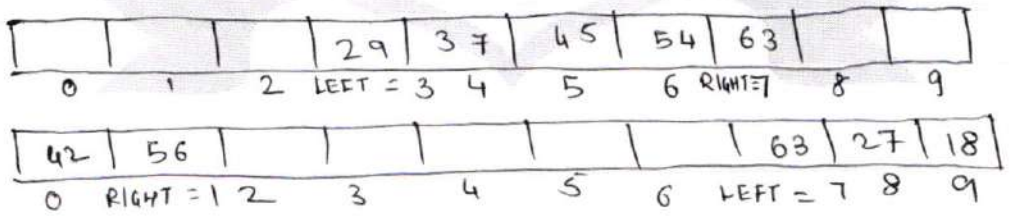
```
4. Exit
Enter your choice: 3

Circular Queue Elements are :
10      20      30      40      50
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted element : 10

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1

Enter the value to be inserted:  60

Insertion Success!!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

Circular Queue Elements are :
20      30      40      50      60      */
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Deques

A deque (pronounced as 'deck' or dequeue') is a list in which the element can be inserted or deleted at either end. It is also known as a head-tail linked list because elements can be added to or removed from either the front (head) or the back (tail) end. However, no element can be added and deleted from middle. In the computer's memory, a deque is implemented using either a circular array or a circular doubly linked list. In a deque, two pointers are maintained, LEFT and RIGHT, which point to either end of the deque. The elements in a deque extend from the LEFT end to the RIGHT end and since it is circular, Dequeue[N-1] is followed by Dequeue[0]. Consider the deque in following fig



Double-ended queue

There are two variants of a double-ended queue. They include

a) Input restricted deque - In this deque, insertions can be done only at one of the ends while deletions can be done from both ends.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

a) Output restricted deque In this dequeue deletions can be done ~~from~~ only at one of the ends while insertions can be done on both ends.

```
/*
Aim:- Implementation of Double Ended Queue menu driven program*/
#include<stdio.h>
#include<stdlib.h>
# define size 40

int front=-1,rear=-1;
int deque[size];
void insert_rear(int max)
{
 int x;
 if((front==0 && rear==max-1)||front==rear+1)
 {
  printf("\nOVERFLOW ! ! !");
 }
 else
 {
  printf("\nEnter the value : ");
  scanf("%d",&x);
  if(front==-1)
    front=rear=0;
  else
  if(rear==max-1)
    rear=0;
  else
   rear++;
  deque[rear]=x;
 }
}

void insert_front(int max)
{
 int x;
 if((front==0 && rear==max-1)||front==rear+1)
 {
  printf("\nOVERFLOW ! ! !");
 }
 else
 {
  printf("\nEnter the value : ");
  scanf("%d",&x);
  if(front==-1)
    front=rear=0;
  else
  if(front==0)
    front=max-1;
  else
   front--;
```

```
 deque[front]=x;
 }
}
void delete_front(int max)
{
 if(front==-1)
 {
  printf("\nUNDERFLOW ! ! !");
 }
 else
 {
  printf("\n%d is deleted....",deque[front]);
  if(front==rear)
    front=rear=-1;
  else
  if(front==max-1)
    front=0;
  else
    front++;
 }
}
void delete_rear(int max)
{
 if(front==-1)
 {
  printf("\nUNDERFLOW ! ! !");
 }
 else
 {
  printf("\n%d is deleted....",deque[rear]);
  if(front==rear)
    front=rear=-1;
  else
  if(rear==0)
    rear=max-1;
  else
    rear--;
 }
}
void display(int max)
{
 int f=front,r=rear;
 if(f==-1)
 {
  printf("\nSORRY ! NO ELEMENT ! ! !");
 }
 else
 {
```

```
printf("\n");
if(f<=r)
{
 while(f<=r)
 {
  printf("%d ",deque[f]);
  f++;
 }
}
else
{
 while(f<=max-1)
 {
  printf("%d ",deque[f]);
  f++;
 }
 f=0;
 while(f<=rear)
 {
  printf("%d ",deque[f]);
  f++;
 }
}
}
void input_deque(int max)
{
 int ch;
 do
 {
  printf("\n-----INPUT RESTRICTED DEQUEUE-----");
  printf("\n1. Insert at rear\n2. Delete from front\n3. Delete from rear\n4.
Display\n5. Exit");
  printf("\n---------------------------------");
  printf("\nEnter your choice : ");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1: insert_rear(max);
           break;
   case 2: delete_front(max);
           break;
   case 3: delete_rear(max);
           break;
   case 4: display(max);
           break;
   case 5: exit(0);
           break;
```

```
  default : printf("\nOOPS ! WRONG INPUT !");
 }
 }while(ch!=5);
}
void output_deque(int max)
{
 int ch;
 do
 {
 printf("\n-----OUTPUT RESTRICTED DEQUEUE-----");
 printf("\n1. Insert at rear\n2. Insert at front\n3. Delete from front\n4. Display\n5.
Exit");
 printf("\n---------------------------------");
 printf("\nEnter your choice : ");
 scanf("%d",&ch);
 switch(ch)
 {
  case 1: insert_rear(max);
          break;
  case 2: insert_front(max);
          break;
  case 3: delete_front(max);
          break;
  case 4: display(max);
          break;
  case 5: exit(0);
          break;
  default : printf("\nOOPS ! WRONG INPUT !");
 }
 }while(ch!=5);
}

void main()
{
 int ch,max;
 printf("Enter the queue capacity : ");
 scanf("%d",&max);
 printf("\n----------MAIN MENU----------\n");
 printf("1. INPUT RESTRICTED DEQUEUE\n2. OUTPUT RESTRICTED
DEQUEUE\n");
 printf("\nEnter your choice : ");
 scanf("%d",&ch);
 switch(ch)
 {
  case 1: input_deque(max);
          break;
  case 2: output_deque(max);
          break;
```

```
 default : printf("\nOPPS ! WRONG INPUT ! ! !");
 }
}
/*
Enter the queue capacity : 3

----------MAIN MENU----------
1. INPUT RESTRICTED DEQUEUE
2. OUTPUT RESTRICTED DEQUEUE

Enter your choice : 1

-----INPUT RESTRICTED DEQUEUE-----
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
---------------------------------
Enter your choice : 4

10 20 30
-----INPUT RESTRICTED DEQUEUE-----
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
---------------------------------
Enter your choice : 2

10 is deleted....
-----INPUT RESTRICTED DEQUEUE-----
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
---------------------------------
Enter your choice : 3

30 is deleted....
-----OUTPUT RESTRICTED DEQUEUE-----
1. Insert at rear
2. Insert at front
3. Delete from front
4. Display
5. Exit
```

```
---------------------------------
Enter your choice : 4

10 20
-----OUTPUT RESTRICTED DEQUEUE-----
1. Insert at rear
2. Insert at front
3. Delete from front
4. Display
5. Exit
---------------------------------
Enter your choice : 1

Enter the value : 30

-----OUTPUT RESTRICTED DEQUEUE-----
1. Insert at rear
2. Insert at front
3. Delete from front
4. Display
5. Exit
---------------------------------
Enter your choice : 4

10 20 30
-----OUTPUT RESTRICTED DEQUEUE-----
1. Insert at rear
2. Insert at front
3. Delete from front
4. Display
5. Exit
---------------------------------
Enter your choice : 2

Enter the value : 9


Enter your choice : 4

9 10 20 30

Enter your choice : 3

9 is deleted....
*/
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Priority Queues

A priority queue is a data structure in which each element is assigned a priority. The priority of the element will be used to determine the order in which the elements will be processed. The general rules of processing the element of a priority queue are

- An element with higher priority is processed before an element with a lower priority
- Two elements with the same priority are processed on a first-come-first-served (FCFS) basis

A priority queue can be thought of as a modified queue in which when an element has to be removed from the queue, the one with the highest priority is retrieved first. The priority of the element can be set based on various factors. Priority queues are widely used in operating systems to execute the highest priority process first. The priority of the process may be set based on the CPU time it requires to get executed completely for example, if there are three processes where the first process needs 5 ns to complete the second process needs 4ns and the third process needs 7 ns then the second process will have the hightest priority and will thus be the first to be executed

However, CPU time is not the only factor that determines the priority, rather it is just one among several factors. Another factor is the importance of one process over another.

Implementation of a Priority Queue

There are two ways to implement a priority queue. We can either use a sorted list to store the elements so that when an element has to be taken out, the queue will not have to be searched for the element with the highest priority or we can use an unsorted list so that insertions are always done at the end of the list. Every time when an element has to be removed from from List, the element with highest priority will be searched and removed. While a sorted list takes O(n) time to insert an element in the list it takes only O(1) time to delete an element.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Application of Queues

- Queue are widely used as waiting lists for a single shared resource like printer, disk, CPU
- Queue are used to transfer data asynchronously (data not necessarily received at same rate as sent) between two process (IO buffers) e.g pipes, file Io, sockets.
- Queue are used as buffers on MP3 player and portable CD player, iPod playlist
- Queue are used in Playlist for jukebox to add songs to the end, play from the front of the list.
- Queue are used in operating system for handling interrupts. When programming a real-time system that can be interrupted. For example, by a mouse click, it is necessary to process the ~~th~~ interrupts immediately, before proceeding with the current job. If the interrupt have to be handled in order of arrival then a FIFO queue is the appropriate data structure.

Josephus Problem:

Let us see how queue can be used for ~~hand~~ finding a solution to the Josephus problem.

In Josephus problem, n people stand in a circle waiting to be executed. The

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

counting starts at some point in the circle and proceeds in a specific direction around the circle. In each step, a certain number of people are skipped and the next person is executed. The elimination of people makes the circle smaller and smaller. At the last step, only one person remains who is declared the winner.

Therefore, if there are n number of people and a number k which indicates that k-1 people are skipped and k-th person in the circle is eliminated, then the problem is to choose a position in the initial circle so that the given person becomes the winner.

For example, if there are 5 (n) people and every second (k) person is eliminated then first the person at position 2 is eliminated followed by the person at position 4 followed by person at position 1 and finally the person at position 5 is eliminated. Therefore, the person at position 3 becomes the winner.