# Experiment No. 4

Aim:

Queues are frequently used in computer programming & a typical example is the creation of a job queue by an operating system. If the operating system doesnot use priorities, then jobs are processed in order they enter the system. Write C++ program for stimulating job queue. Write functions to add job and delete job from queue.

Pre-requisite:

Basics of Queue
Different operations that can be performed on queue.

Objectives:

To perform addition & deletion operation on queue.

Input:

Size of queue. Element in queue.

Outcome:

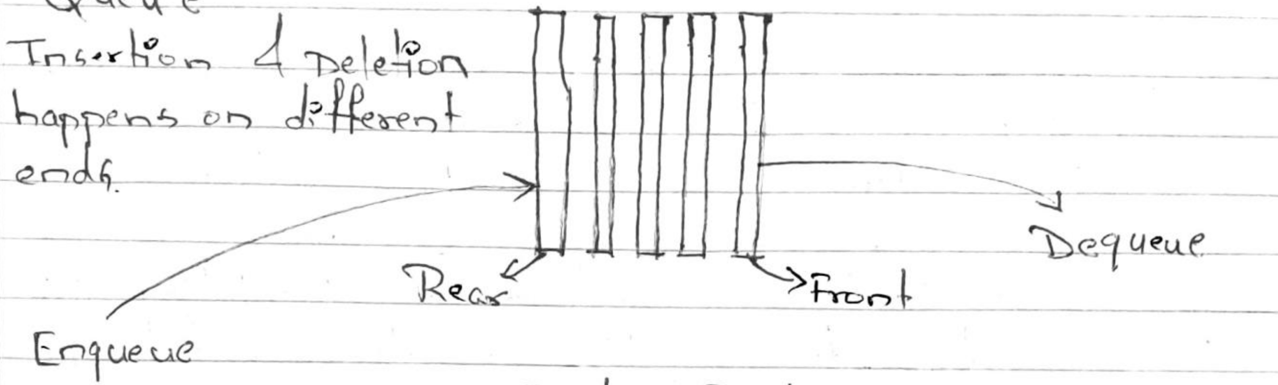
Result of addition of job operation on queue.
Result of deletion of job operation on queue.

Theory:

A queue is a linear structure which follows a particular order in which operations are performed. The order in First In First out (FIFO). A good example of a queue is any queue of consumer for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In stack we remove the item the most recently added. In a queue we reomove the item the least recently added.

Queue

Insertion & Deletion happens on different ends.



First in First out.

Applications of Queue Data Structure:

Queue is used when things don't have to be processed immediately, but have to be processed in FIFO order like First search. This property of Queue makes it also useful in following kind of senarious.

1). When a resource is shared among multiple consumers. Examples includes CPU scheduling Disk scheddling.

2) When data is transfered asynchronowly between two Processes. Example includes IO Buffers, pipes, file IO, etc.

3) In Operating systems:

a) Semaphores

b) FCFS (first come first service) scheduling, example: FIFO queue

c) Spooling in printers.

d) Buffer for devices like keyboards.

4) In Networks:-

a) Queues in routers / switches

b) Mail Queues.

5) Variations:

Dequeue, Priority Queue, Doubly ended priority queue.

## Basic Operations:

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

Enqueue() - add (store) an item to queue.

Dequeue() - remove (access) an item from queue.

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

peek() - gets the element at front of queue without removing it.

isfull() - check if queue is full.

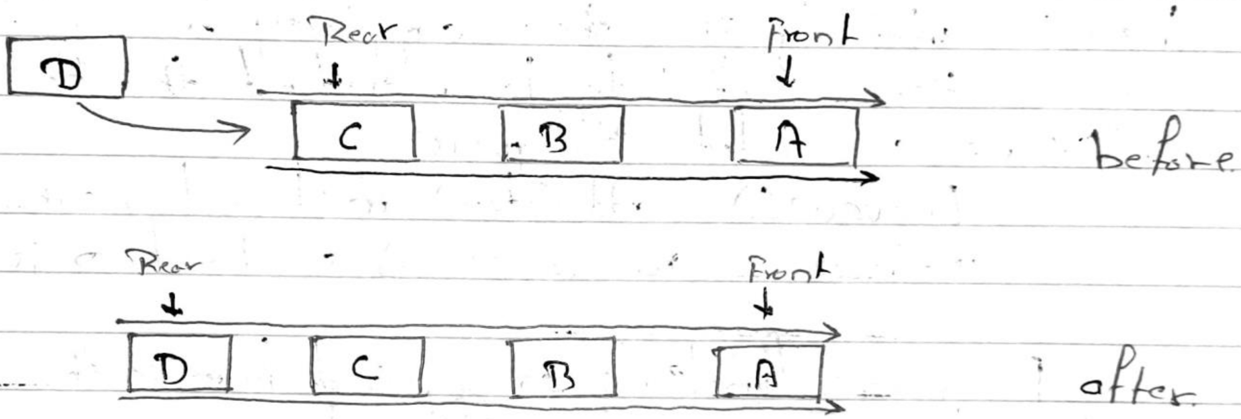
is empty() - check if queue is empty.

## Enqueue Operation:

Queues maintain two data pointers front & rear. Therefore, its operation are comparitavely difficult to implement that than of stacks.

The following steps should be taken to enquieue (insert) data into a queue.

Step I: Check if queue is full.
Step II: If queue is full, produce overflow error & exit.
Step III: If queue is not full, increment rear pointer to point the next empty space.
Step IV: Add data element to queue location, where rear is pointing
Step V: Return success.

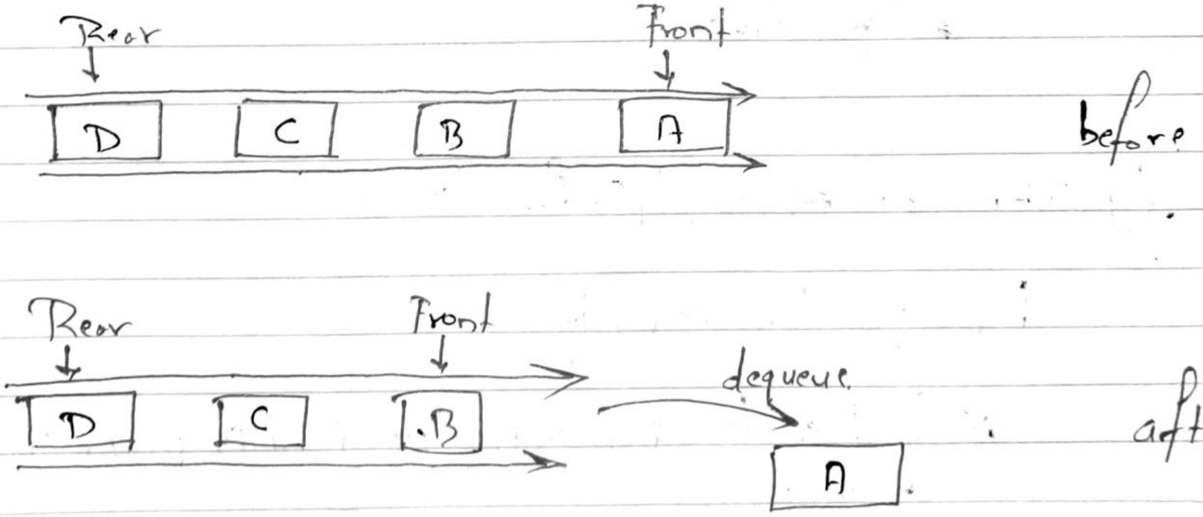


Queue Enqueue.

## Dequeue Operation:

Accessing data from 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 I: Check if queue is empty.
Step II: If queue is empty, produce underflow error & exit.
Step III: If queue is not empty, access data where front is pointer.
Step IV: Increment front pointer to point the next available data element.
Step V: Return success.



Queue dequeue.

## Algorithm of peek() function:

```
begin procedure peek
  return queue [front]
end procedure
```

## Algorithm of isfull() function:

```
begin procedure isfull

  if rear equals to MAXSIZE
     return true
  else
     return false
  endif
     end procedure.
```

## Algorithm of isempty() function:

```
begin procedure isempty

  if front is less than MIN OR front is greater than rear
     return true
  else
     return false
  endif
     end procedure
```

Algorithm of enqueue() operation:

```
procedure enqueue(data)

   if queue is full
      return overflow
   endif
         rear ← rear + 1
      queue [rear] ← data
      return true

end procedure
```

Algorithm of dequeue() operation:

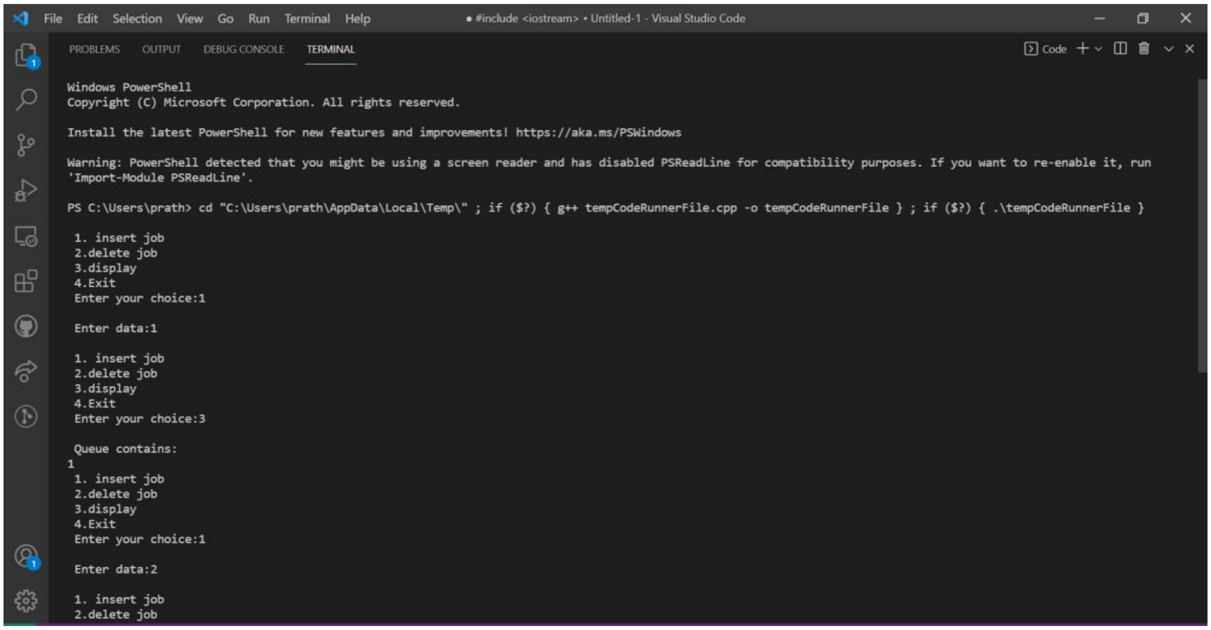
```
procedure dequeue

   if queue is empty
      return underflow
   endif
      data = queue[front]
      front ← front + 1
      return true

end procedure
```

## Program Code:

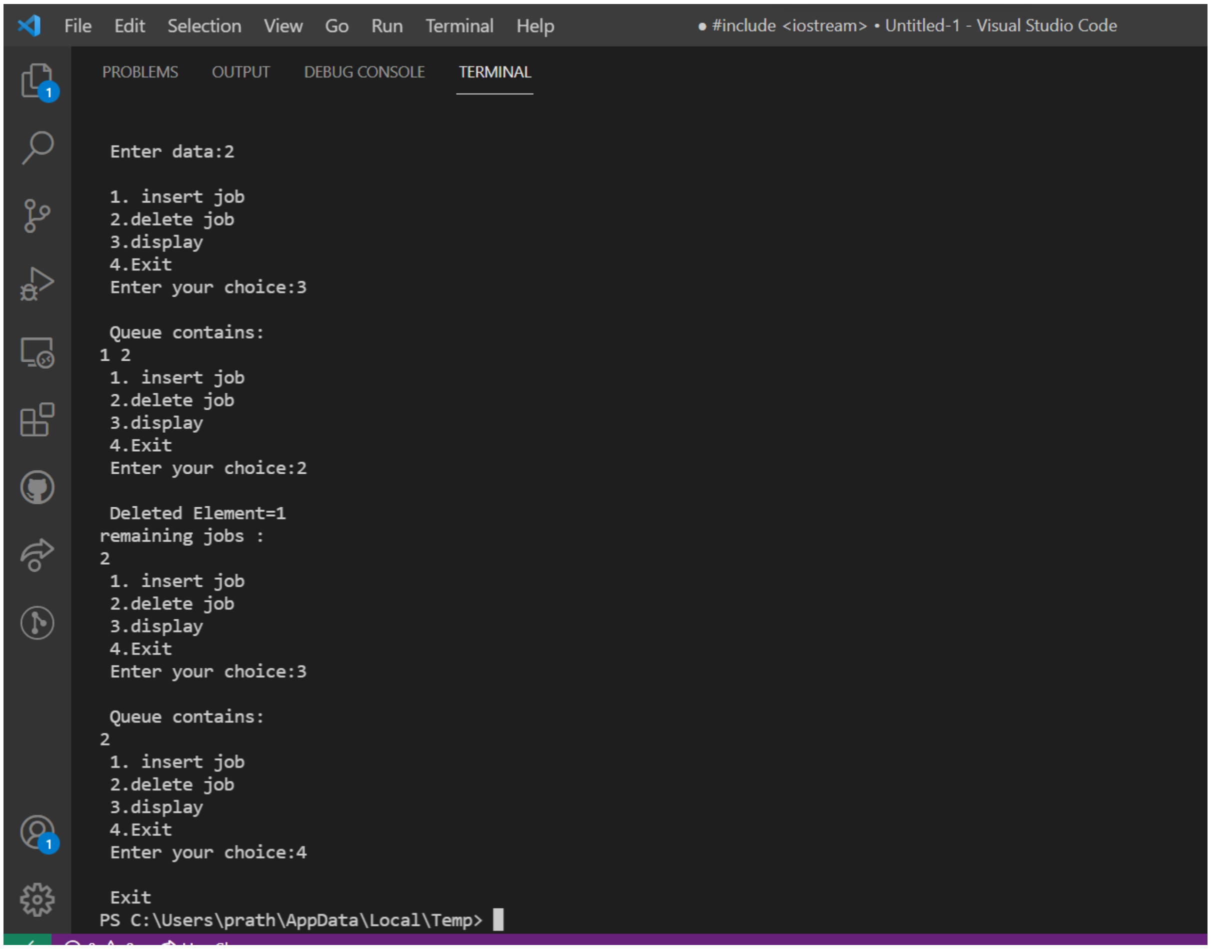
```
#include <iostream>
#define MAX 10
using namespace std;
struct queue
{       int data[MAX];
            int front,rear;
};
class Queue
{    struct queue q;
   public:
      Queue(){q.front=q.rear=-1;}
      int isempty();
      int isfull();
      void enqueue(int);
      int delqueue();
      void display();
};
int Queue::isempty()
{
            return(q.front==q.rear)?1:0;
}
int Queue::isfull()
{    return(q.rear==MAX-1)?1:0;}
void Queue::enqueue(int x)
{q.data[++q.rear]=x;}
int Queue::delqueue()
{return q.data[++q.front];}
void Queue::display()
{   int i;
    cout<<"\n";
    for(i=q.front+1;i<=q.rear;i++)
                 cout<<q.data[i]<<" ";
}
int main()
{
     Queue obj;
            int ch,x;
            do{    cout<<"\n 1. insert job\n 2.delete job\n 3.display\n 4.Exit\n Enter your
choice:";
                   cin>>ch;
            switch(ch)
            {  case 1: if (!obj.isfull())
                           {   cout<<"\n Enter data:";
                                 cin>>x;
                                 obj.enqueue(x);
                           }
```

```
                    else
                           cout<< "Queue is overflow";
                     break;
              case 2: if(!obj.isempty())
                                     cout<<"\n Deleted Element="<<obj.delqueue();
                          else
                                 {   cout<<"\n Queue is underflow";  }
                          cout<<"\nremaining jobs :";
                          obj.display();
                     break;
             case 3: if (!obj.isempty())
                  {  cout<<"\n Queue contains:";
                             obj.display();
                  }
                  else
                              cout<<"\n Queue is empty";
                 break;
             case 4: cout<<"\n Exit";
        }
      }while(ch!=4);
return 0;
}
```

## Program Output:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run
'Import-Module PSReadLine'.

PS C:\Users\prath> cd "C:\Users\prath\AppData\Local\Temp\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:1

 Enter data:1

 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:3

 Queue contains:
1
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:1

 Enter data:2

 1. insert job
 2.delete job
```



```
 Enter data:2

 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:3

 Queue contains:
1 2
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:2

 Deleted Element=1
remaining jobs :
2
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:3

 Queue contains:
2
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:4

 Exit
PS C:\Users\prath\AppData\Local\Temp> 
```

Conclusion:

By this way we can perform different operations on queue.