Queue

Experiment No.:1b

06/08/2012

# AIM:

# Implementation of a Queue.

# ALGORITHM:

The structure of a queue is as following :

|  |  |  |  |
| --- | --- | --- | --- |
| front | rear | length | buffer |

Here front,rear is index of newest and oldest element of the queue which is initialized as -1, length is the length of the stack and buf is the array of data.

The following function checks whether a queue is full or not.

Queue.full() {

if(Queue.rear = Queue.length - 1)

return true;

else

return false;

}

The following function checks whether a queue is empty or not.

Queue.empty() {

if(Queue.rear=-1)

return true

else

return false

}

The following function inserts an element e at the front of queue Queue

Queue.insert(e) {

if(NOT Queue.full()){

Queue.rear = Queue.rear + 1

Queue.buf[Queue.rear]=e

}

}

The following function deletes an element from rear of the queue an returns if successful.

Queue.delete() {

if(NOT Queue.empty()) {

e = Queue.buf[Queue.front+1]

shift Queue.buf left by 1

Queue.rear = -1

return e

}

}

# SOURCE CODE:

# /\*\*

# \* Author : Rakesh Malik

# \* Date : 24/07/2012

# \* Subject : Generic Queue Class

# \* Assignment no. :

# \*/

# #include <iostream>

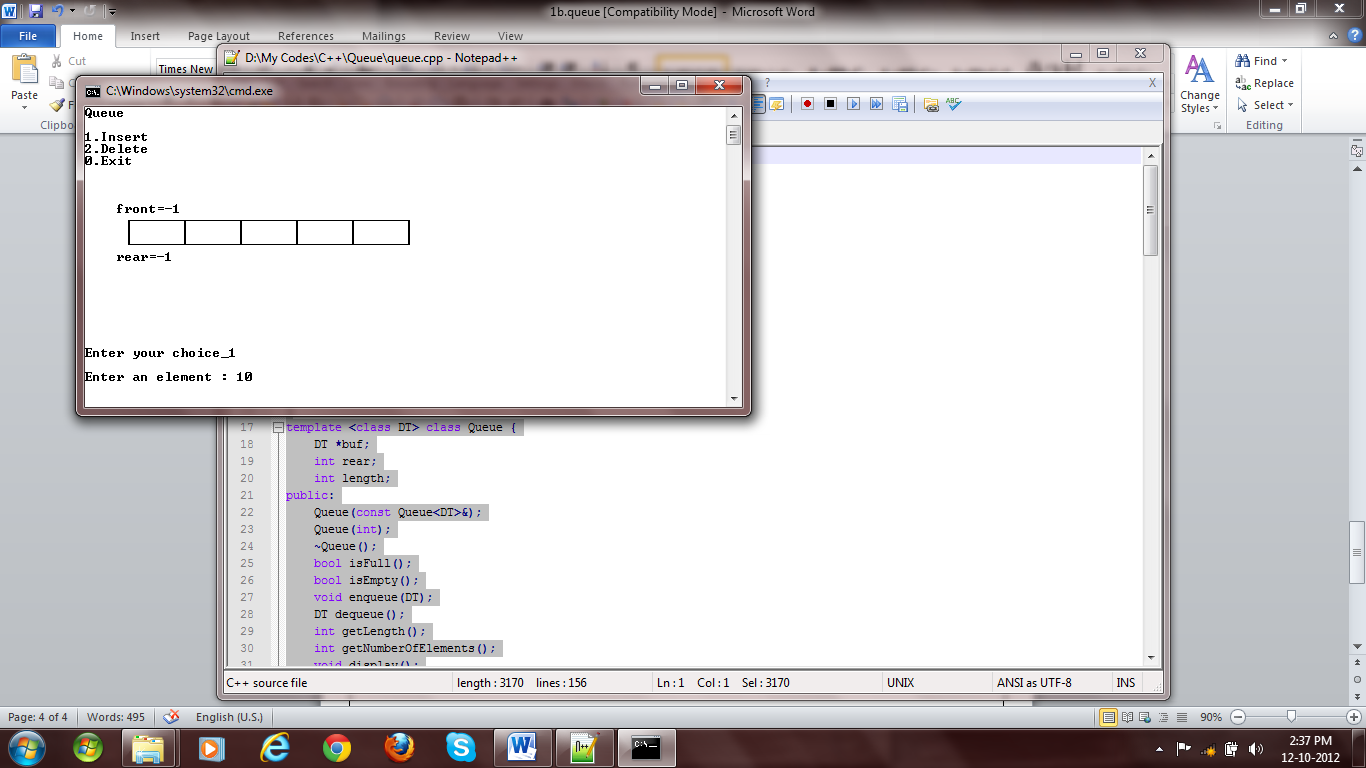


Fig 1: Empty Queue

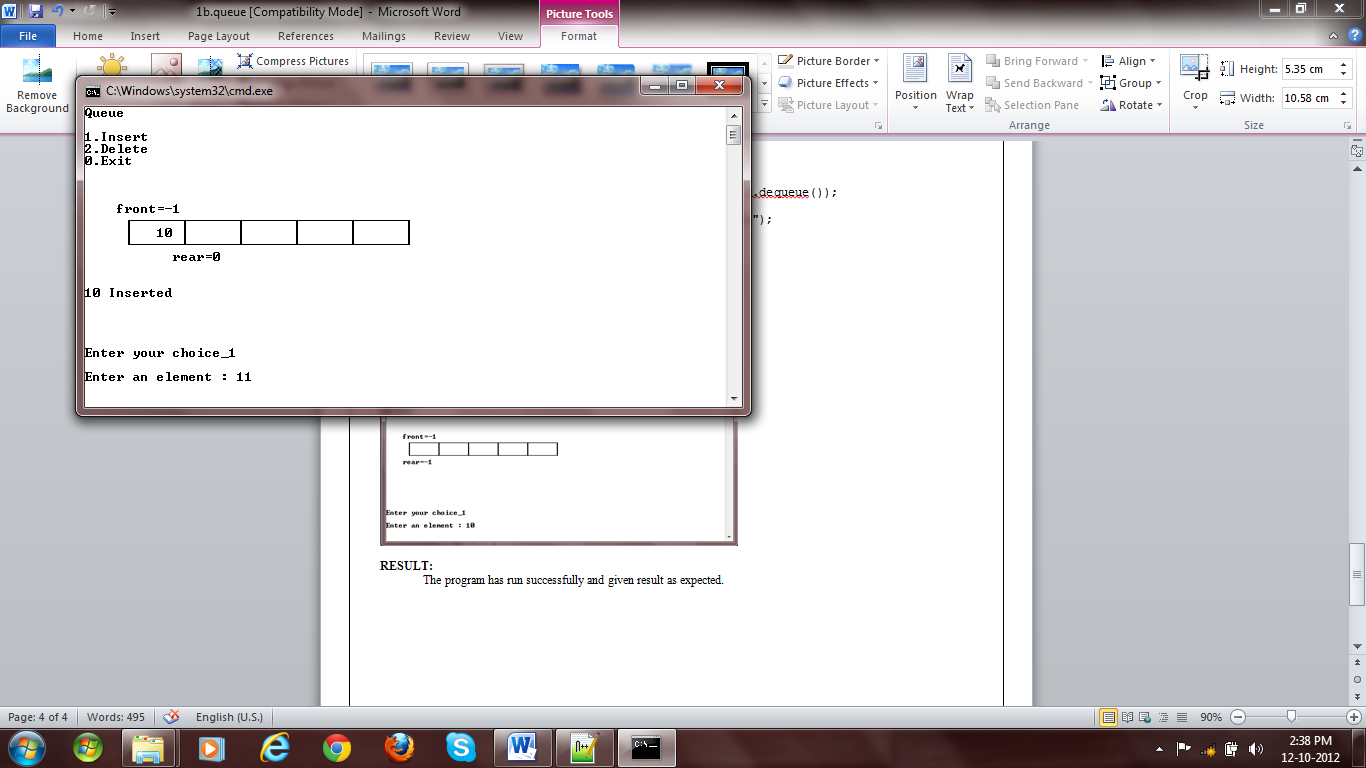


Fig 2:Insertion

# #include <cstdio>

# #include <cstdlib>

# #include "..\myexception.h"

# #include <iomanip>

# using namespace std;

# using namespace exception;

# template <class DT> class Queue {

# DT \*buf;

# int rear;

# int length;

# public:

# Queue(const Queue<DT>&);

# Queue(int);

# ~Queue();

# bool isFull();

# bool isEmpty();

# void enqueue(DT);

# DT dequeue();

# int getLength();

# int getNumberOfElements();

# void display();

# void clear();

# };

# template <class DT> Queue<DT>::Queue(const Queue<DT>& q) {

# Queue::length=q.length;

# Queue::front=q.front;

# Queue::rear=q.rear;

# Queue::buf=new DT[q.length];

# for(int i=0;i<q.length;i++)

# Queue::buf[i]=q.buf[i];

# }

# template <class DT> Queue<DT>::Queue(int length=0) {

# if(length>=0)

# {

# Queue::length=length;

# Queue::buf=new DT[length];

# }

# else

# {

# Queue::length=0;

# Queue::buf=new DT[0];

# }

# Queue::rear=-1;

# }

# template <class DT> Queue<DT>::~Queue() {

# delete Queue::buf;

# }

# template <class DT> bool Queue<DT>::isFull() {

# return Queue::rear==Queue::length-1;

# }

# template <class DT> bool Queue<DT>::isEmpty() {

# return Queue::rear==-1;

# }

# template <class DT> void Queue<DT>::enqueue(DT e) {

# if(!Queue::isFull()){

# Queue::rear++;

# Queue::buf[Queue::rear]=e;

# }

# else

# throw QueueFullException();

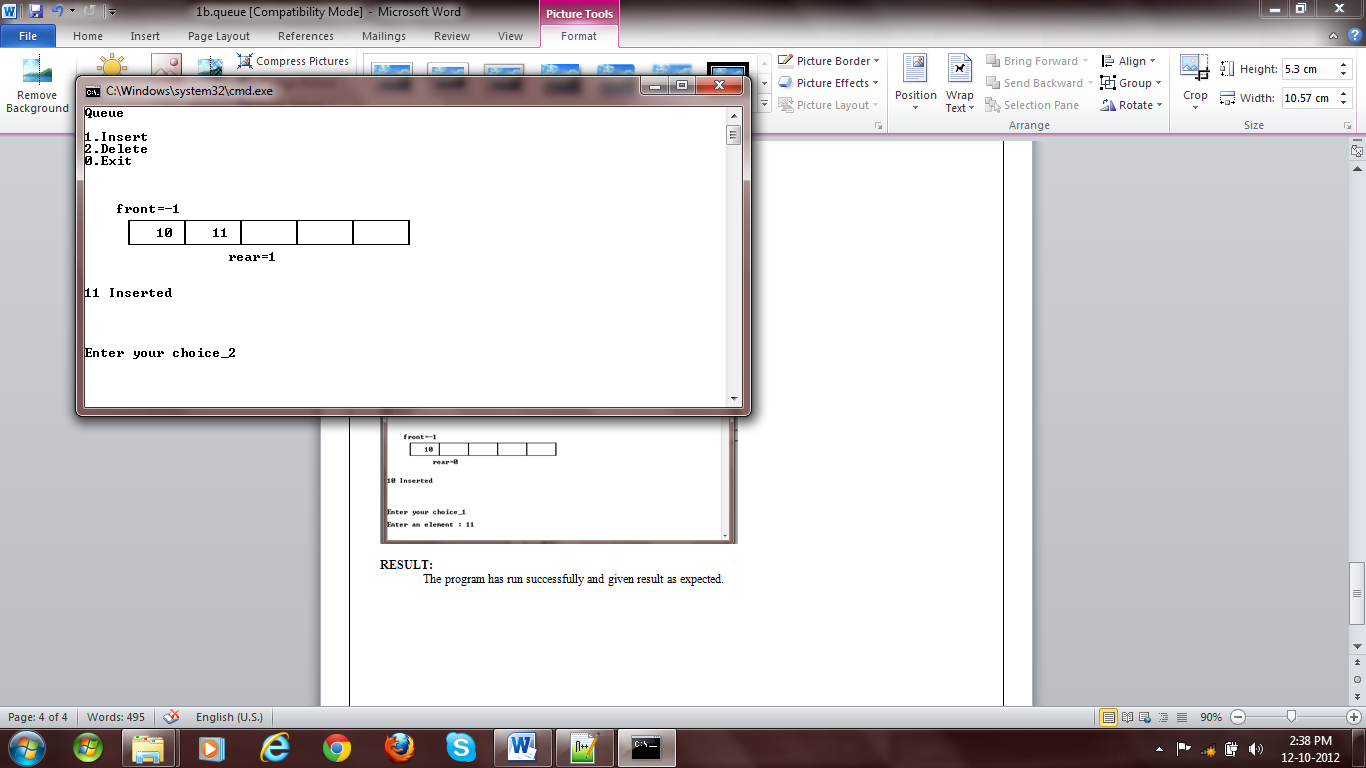


Fig 3: Insertion

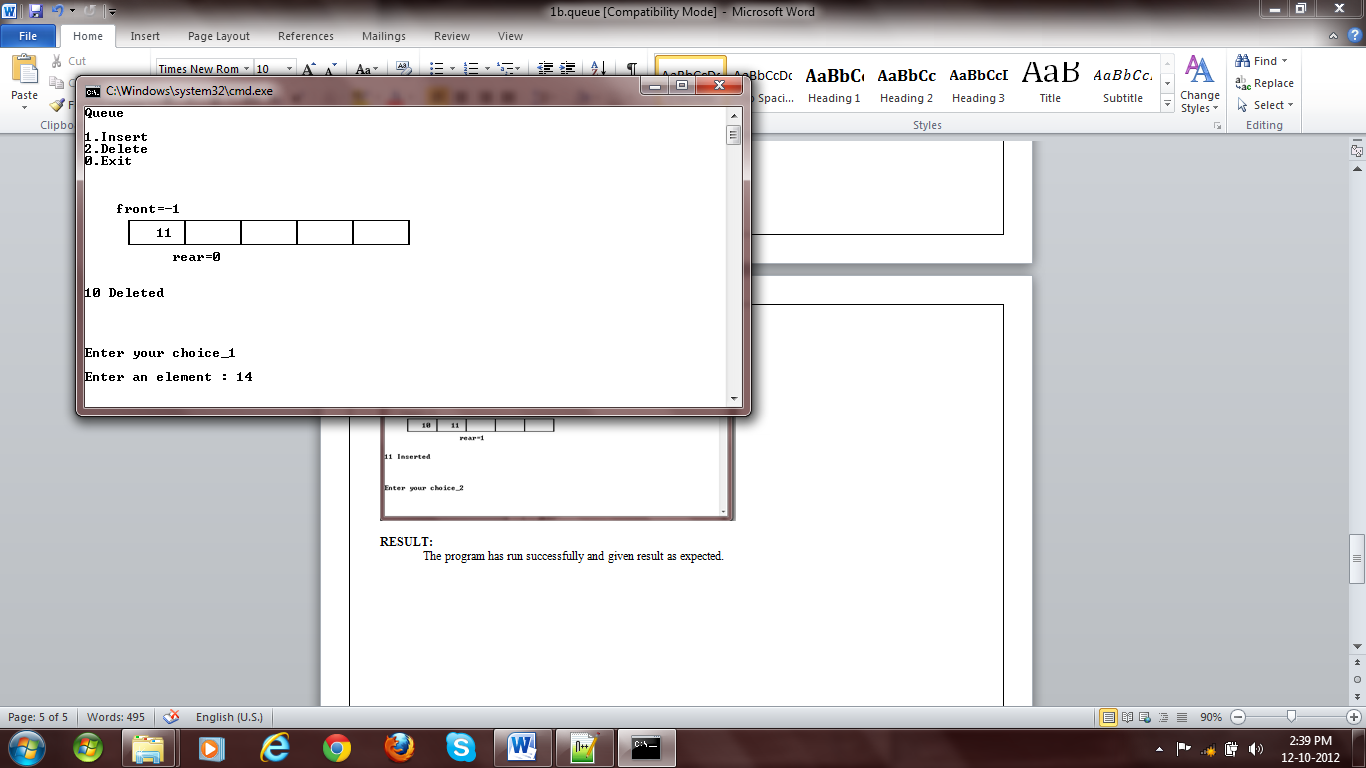


Fig 4: Deletion

# }

# template <class DT> DT Queue<DT>::dequeue() {

# if(!Queue::isEmpty())

# {

# int e=Queue::buf[0];

# for(int i=0; i<Queue::rear; i++)

# Queue::buf[i]=Queue::buf[i+1];

# Queue::rear--;

# return e;

# }

# else

# throw QueueEmptyException();

# }

# template <class DT> int Queue<DT>::getLength() {

# return Queue::length;

# }

# template <class DT> int Queue<DT>::getNumberOfElements() {

# return Queue::rear+1;

# }

# template <class DT> void Queue<DT>::clear() {

# Queue::rear=-1;

# }

# template <class DT> void Queue<DT>::display() {

# cout<<" front=-1"<<endl;

# cout<<" \xDA";

# for(int i=0; i<length-1; i++)

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xC2";

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xBF"<<endl

# <<" ";

# for(int i=0; i<=rear; i++)

# cout<<"\xB3"<<setw(5)<<buf[i]<<" ";

# for(int i=rear+1; i<length; i++)

# cout<<"\xB3 ";

# cout<<"\xB3"<<endl

# <<" \xC0";

# for(int i=0; i<length-1; i++)

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xC1";

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xD9"<<endl;

# for(int i=0; i<=rear; i++)

# cout<<" ";

# cout<<" rear="<<rear<<endl;

# }

# main() {

# Queue<int> q(5);

# int choice=1,e;

# char msg[50]="";

# while(choice) {

# system("cls");

# cout<<"Queue"<<endl

# <<endl

# <<"1.Insert"<<endl

# <<"2.Delete"<<endl

# <<"0.Exit"<<endl

# <<endl<<endl<<endl;

# q.display();

# cout<<endl<<endl

# <<msg<<endl<<endl<<endl<<endl<<endl

# <<"Enter your choice\_";

# cin>>choice;

# cout<<endl;

# switch(choice) {

# case 0:

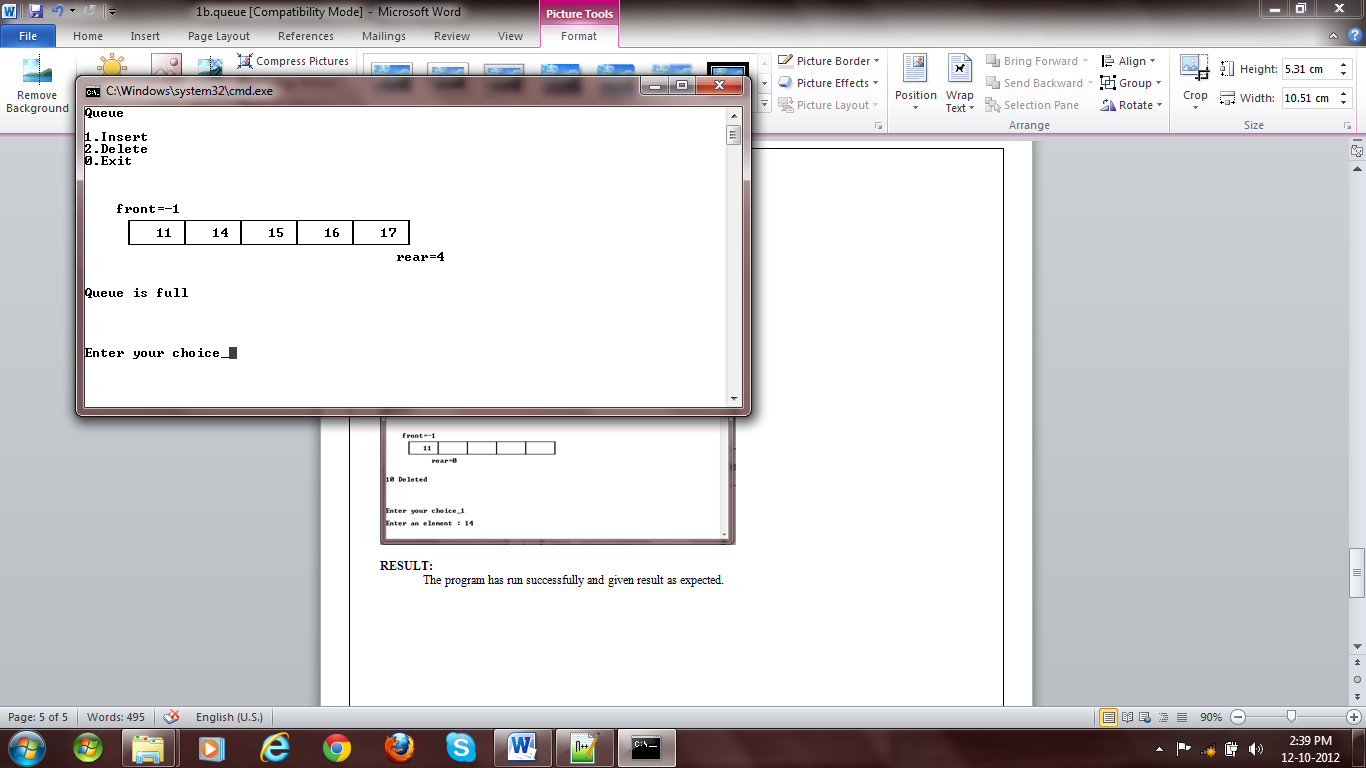


Fig 5: Queue full message while inserting

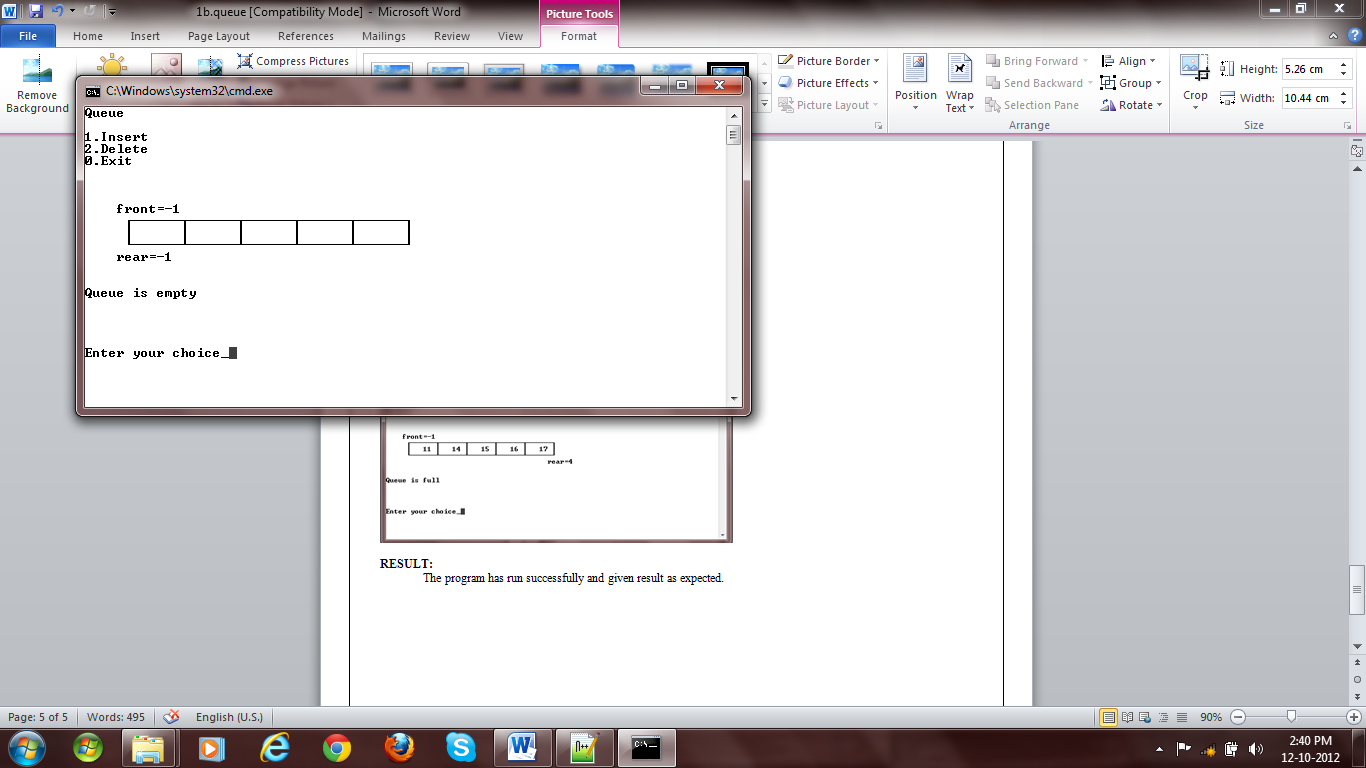


Fig 6: Queue empty message while popping

# return 0;

# case 1:

# if(q.isFull())

# sprintf(msg, "Queue is full");

# else {

# cout<<"Enter an element : ";

# cin>>e;

# q.enqueue(e);

# sprintf(msg, "%d Inserted", e);

# }

# break;

# case 2:

# try {

# sprintf(msg, "%d Deleted", q.dequeue());

# }catch(QueueEmptyException ex) {

# sprintf(msg, "Queue is empty");

# }

# break;

# default:

# sprintf(msg, "Wrong choice");

# }

# }

# }

# RESULT:

The program has run successfully and given result as expected.