Circular Queue

Experiment No.:3a

06/08/2012

# AIM:

# Implementation of a Circular Queue.

# ALGORITHM:

The structure of a queue is as following :

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

Here rear,front 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!=-1 AND (Queue.rear mod Queue.length)=(Queue.front+Queue.length)

mod Queue.length)

return true;

else

return false;

}

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

Queue.empty() {

if(Queue.rear=-1 AND Queue.front=-1)

return true

else

returb false

}

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

Queue.insert(e) {

if(NOT Queue.full()){

Queue.rear=(Queue.rear+1) mod Queue.length

Queue.buf[Queue.rear]=e

}

}

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

Queue.delete() {

if(NOT Queue.empty()) {

Queue.front=(Queue.front+1) mod Queue.length

e=Queue.buf[Queue.front]

if(Queue.front=Queue.rear) {

Queue.front=-1

Queue.rear=-1

}

return e

}

}

# SOURCE CODE:

# /\*\*

# \* Author : Rakesh Malik

# \* Date : 24/07/2012

# \* Subject : Generic Circular Queue Class

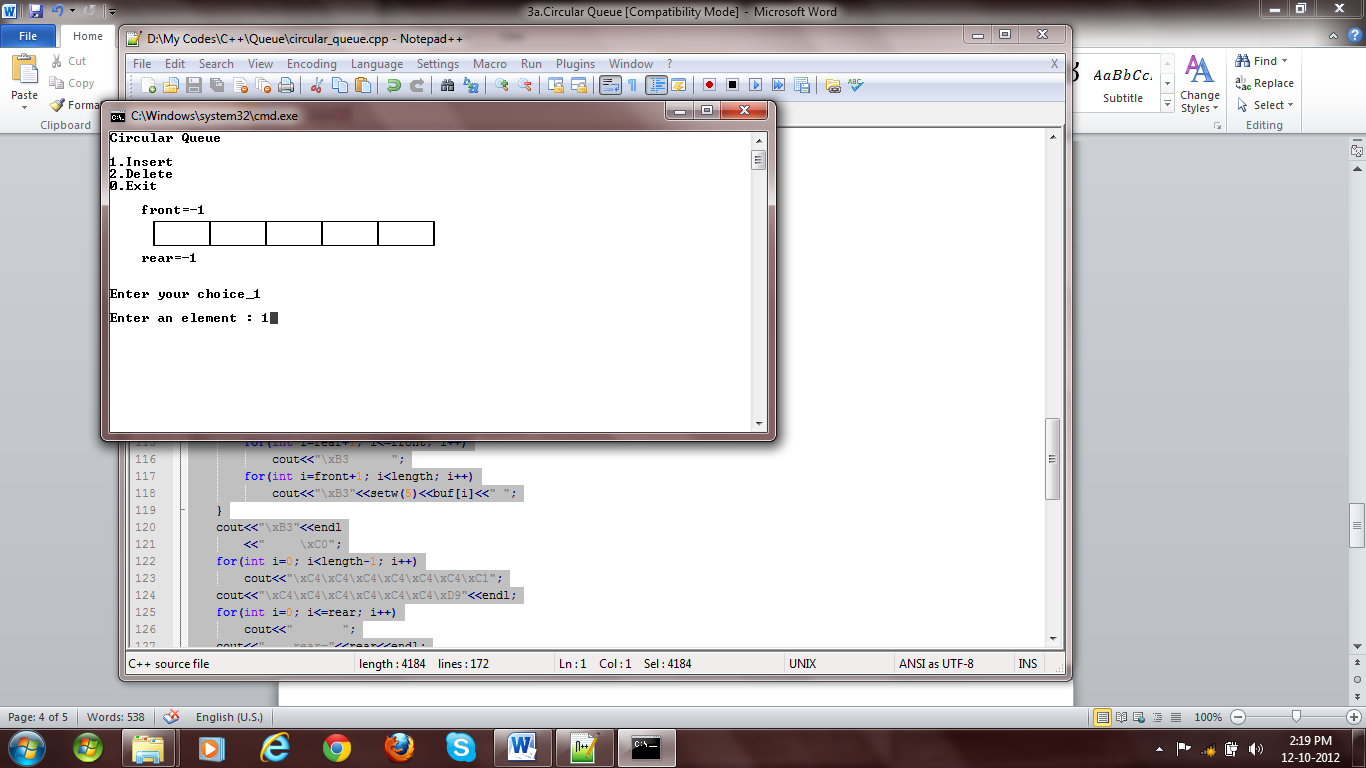


Fig 1: Empty Queue

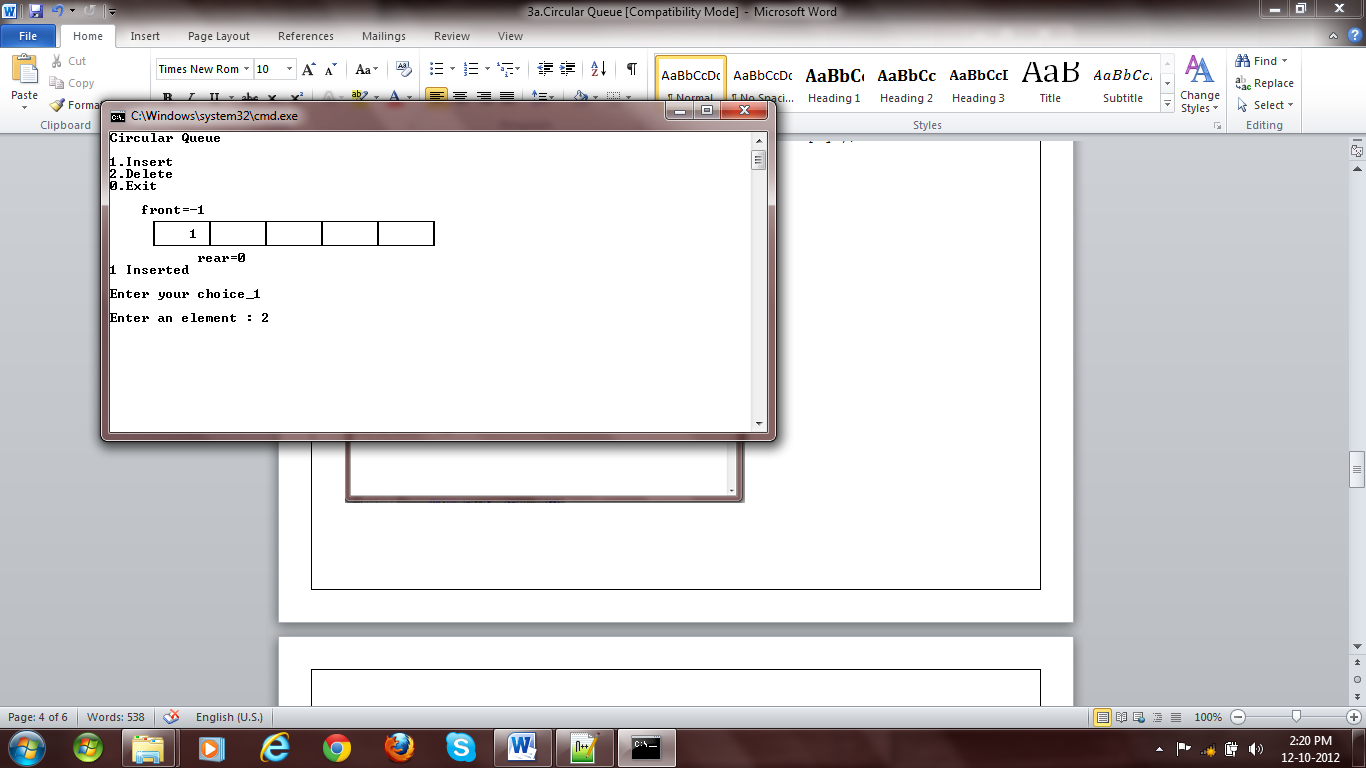


Fig 2: Insertion

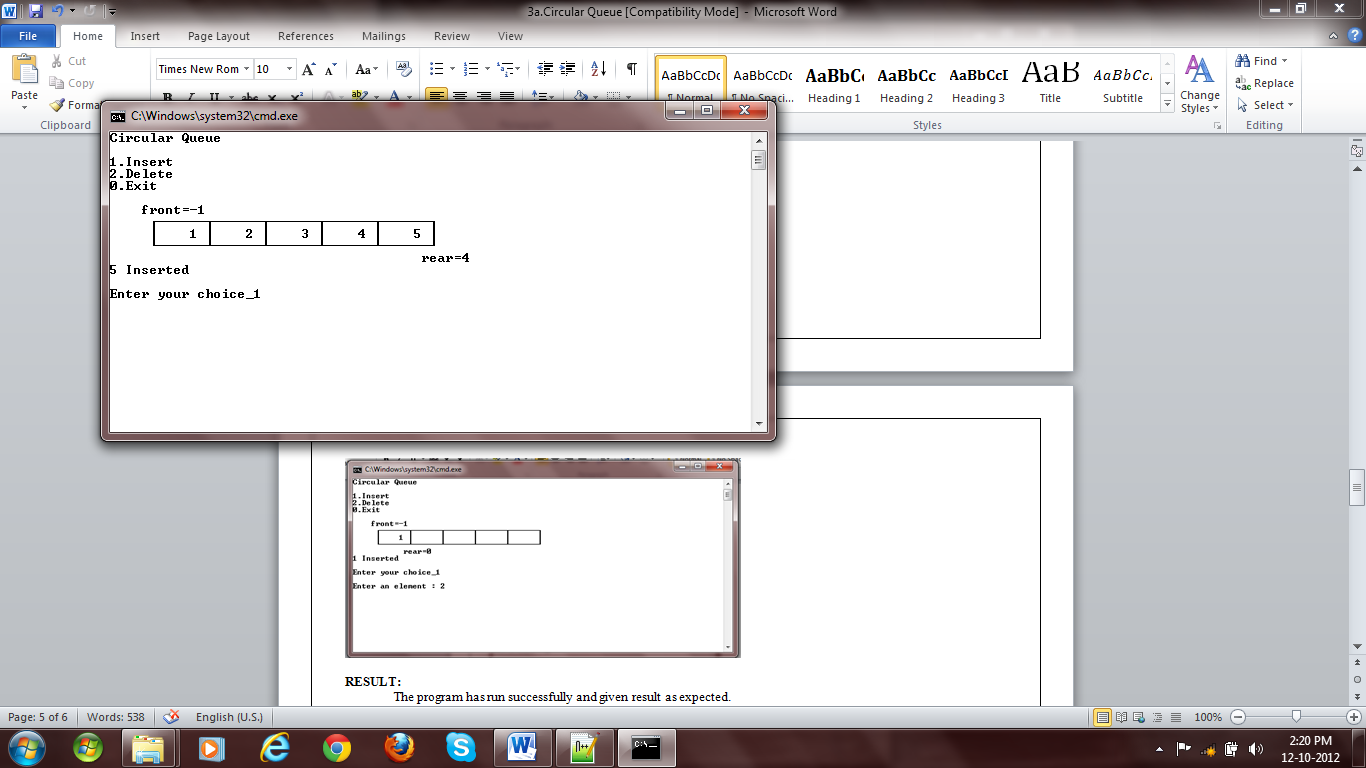


Fig 3: Insertion

# \* Assignment no. :

# \*/

# #include <iostream>

# #include <cstdio>

# #include <iomanip>

# #include <cstdlib>

# #include "..\myexception.h"

# using namespace std;

# using namespace exception;

# template <class DT> class CircularQueue {

# DT \*buf;

# int front,rear;

# int length;

# public:

# CircularQueue(const CircularQueue<DT>&);

# CircularQueue(int);

# ~CircularQueue();

# bool isFull();

# bool isEmpty();

# void enqueue(DT);

# DT dequeue();

# int getLength();

# int getNumberOfElements();

# void display();

# void clear();

# };

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

# CircularQueue::length=q.length;

# CircularQueue::front=q.front;

# CircularQueue::rear=q.rear;

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

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

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

# }

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

# if(length>=0)

# {

# CircularQueue::length=length;

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

# }

# else

# {

# CircularQueue::length=0;

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

# }

# CircularQueue::front=-1;

# CircularQueue::rear=-1;

# }

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

# delete CircularQueue::buf;

# }

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

# return CircularQueue::rear!=-1 && CircularQueue::rear%CircularQueue::length==(CircularQueue::front+CircularQueue::length)%CircularQueue::length;

# }

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

# return CircularQueue::rear==-1 && CircularQueue::front==-1;

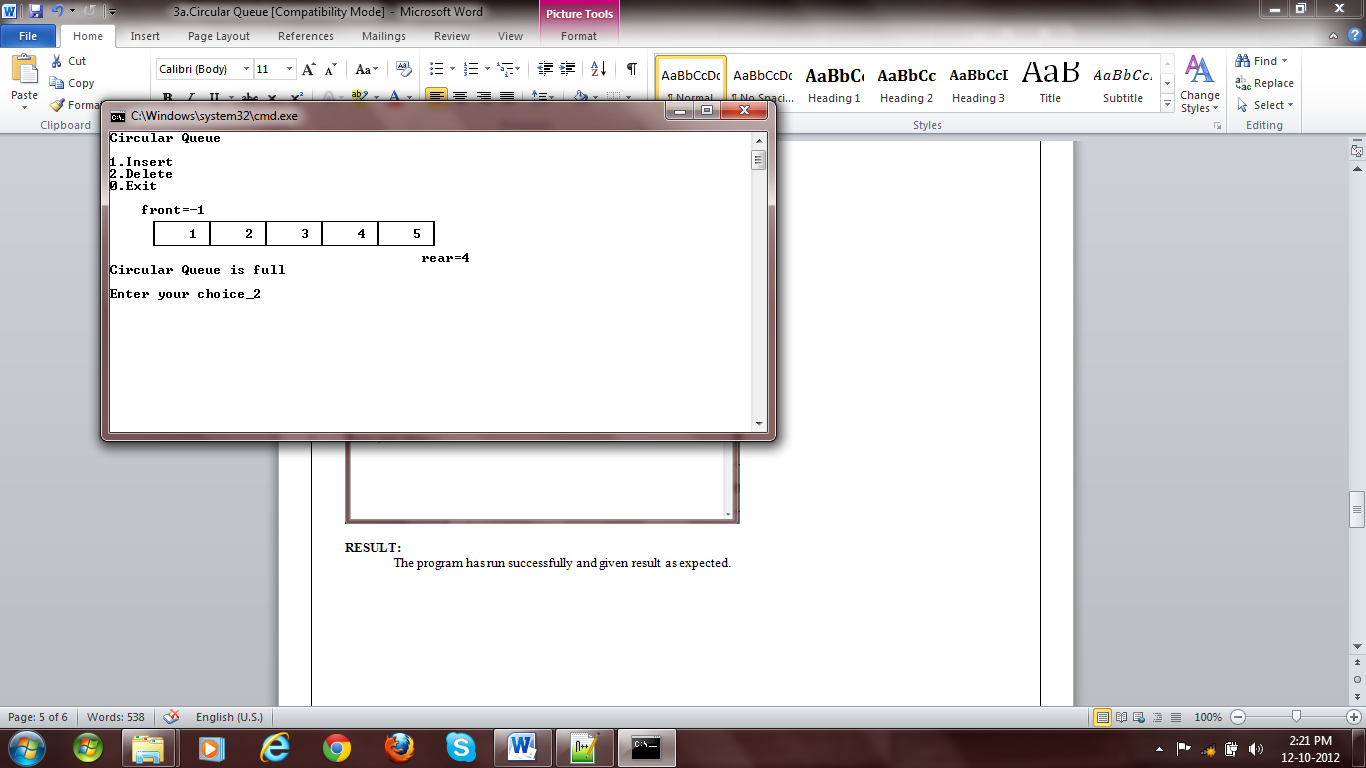


Fig 4: Queue full message while insertion

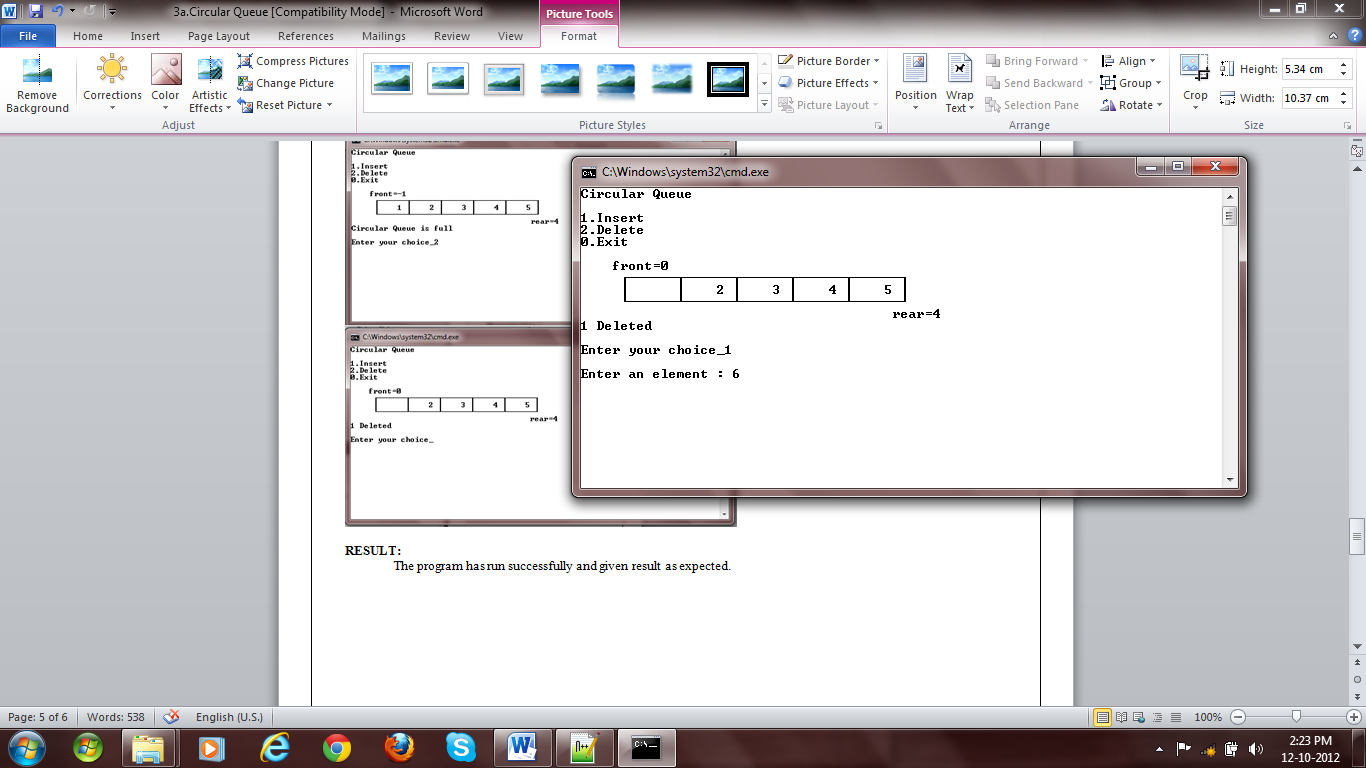


Fig 5: Deletion

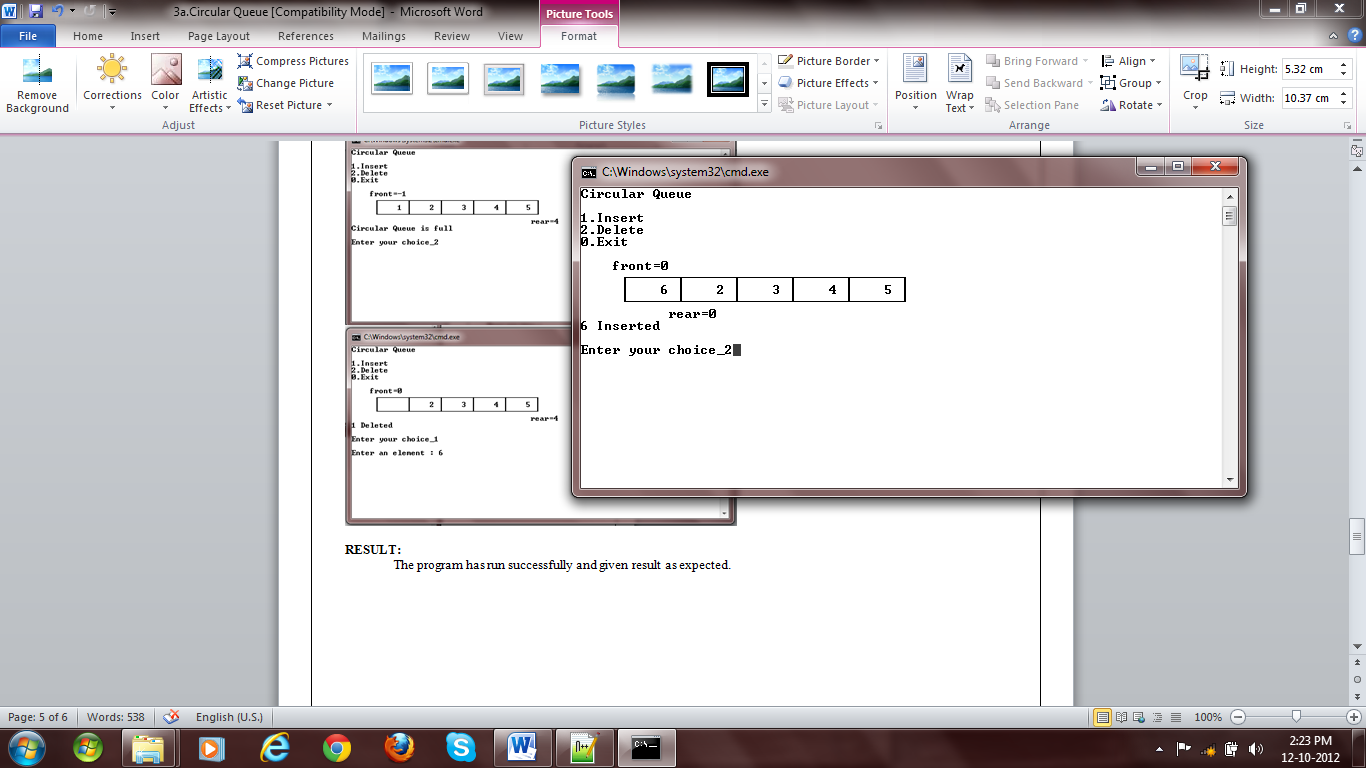


Fig 6: Insertion

# }

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

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

# CircularQueue::rear=(CircularQueue::rear+1)%CircularQueue::length;

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

# }

# else

# throw QueueFullException();

# }

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

# if(!CircularQueue::isEmpty())

# {

# CircularQueue::front=(CircularQueue::front+1)%CircularQueue::length;

# DT e=CircularQueue::buf[CircularQueue::front];

# if(CircularQueue::front==CircularQueue::rear)

# {

# CircularQueue::front=-1;

# CircularQueue::rear=-1;

# }

# return e;

# }

# else

# throw QueueEmptyException();

# }

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

# return CircularQueue::length;

# }

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

# return CircularQueue::front-CircularQueue::rear;

# }

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

# CircularQueue::front=-1;

# CircularQueue::rear=-1;

# }

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

# cout<<" front="<<front<<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

# <<" ";

# if(front<rear || front==-1) {

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

# cout<<"\xB3 ";

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

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

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

# cout<<"\xB3 ";

# } else {

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

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

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

# cout<<"\xB3 ";

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

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

# }

# 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;

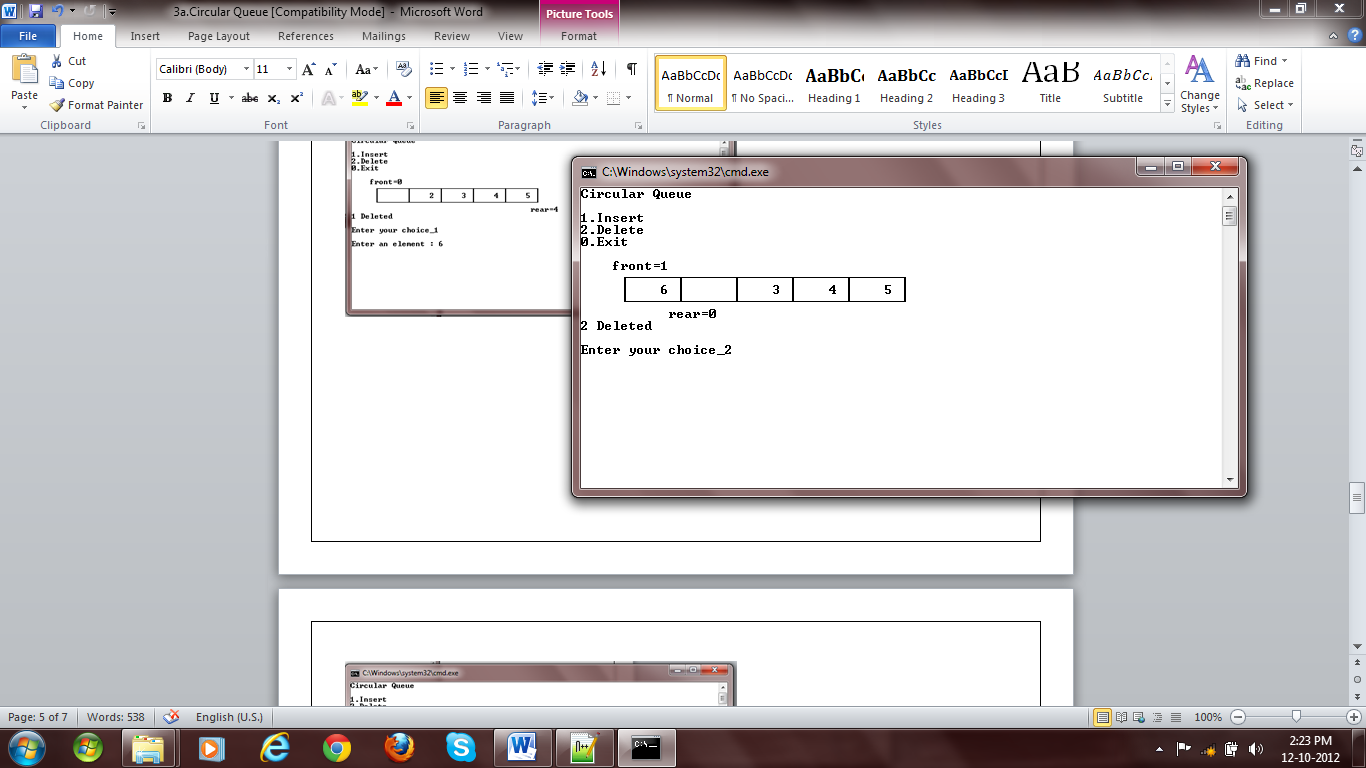


Fig 7: Deletion

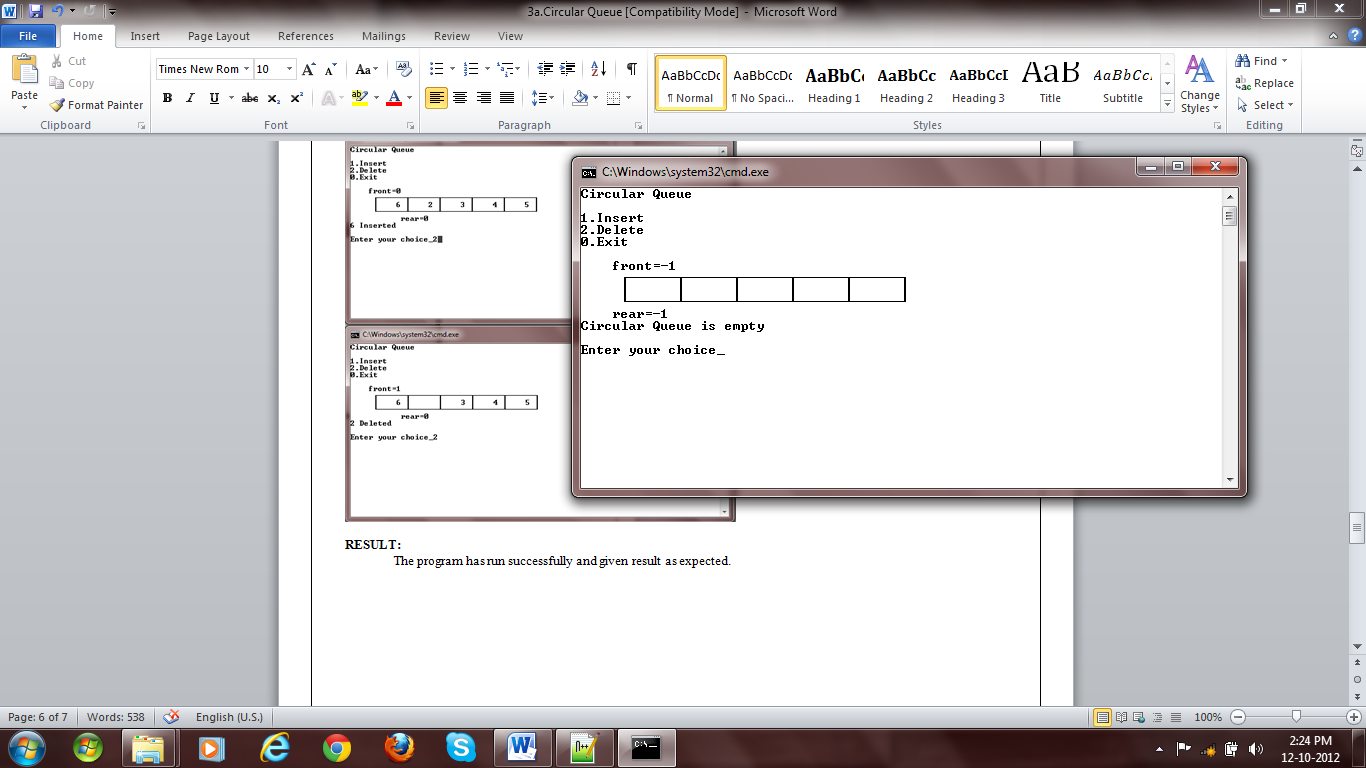


Fig 8: Queue empty message while deleting

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

# cout<<" ";

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

# }

# main() {

# CircularQueue<int> q(5);

# int choice=1,e;

# char msg[50]="";

# while(choice) {

# system("cls");

# cout<<"Circular Queue"<<endl

# <<endl

# <<"1.Insert"<<endl

# <<"2.Delete"<<endl

# <<"0.Exit"<<endl

# <<endl;

# q.display();

# cout<<msg<<endl

# <<endl

# <<"Enter your choice\_";

# cin>>choice;

# cout<<endl;

# switch(choice) {

# case 0:

# return 0;

# case 1:

# if(q.isFull())

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

# }

# break;

# default:

# sprintf(msg, "Wrong choice");

# }

# }

# }

# RESULT:

The program has run successfully and given result as expected.