06/08/2012

Experiment No.:3b

Priority Queue

# AIM:

# Implementation of a Priority 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

Sort (Queue.buf, Queue.rear+1) /\* sort(arr, n) function sorts first n

\* elements of array arr \*/

}

}

The following function deletes an element from rear of the queue adn 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:

# /\*\*

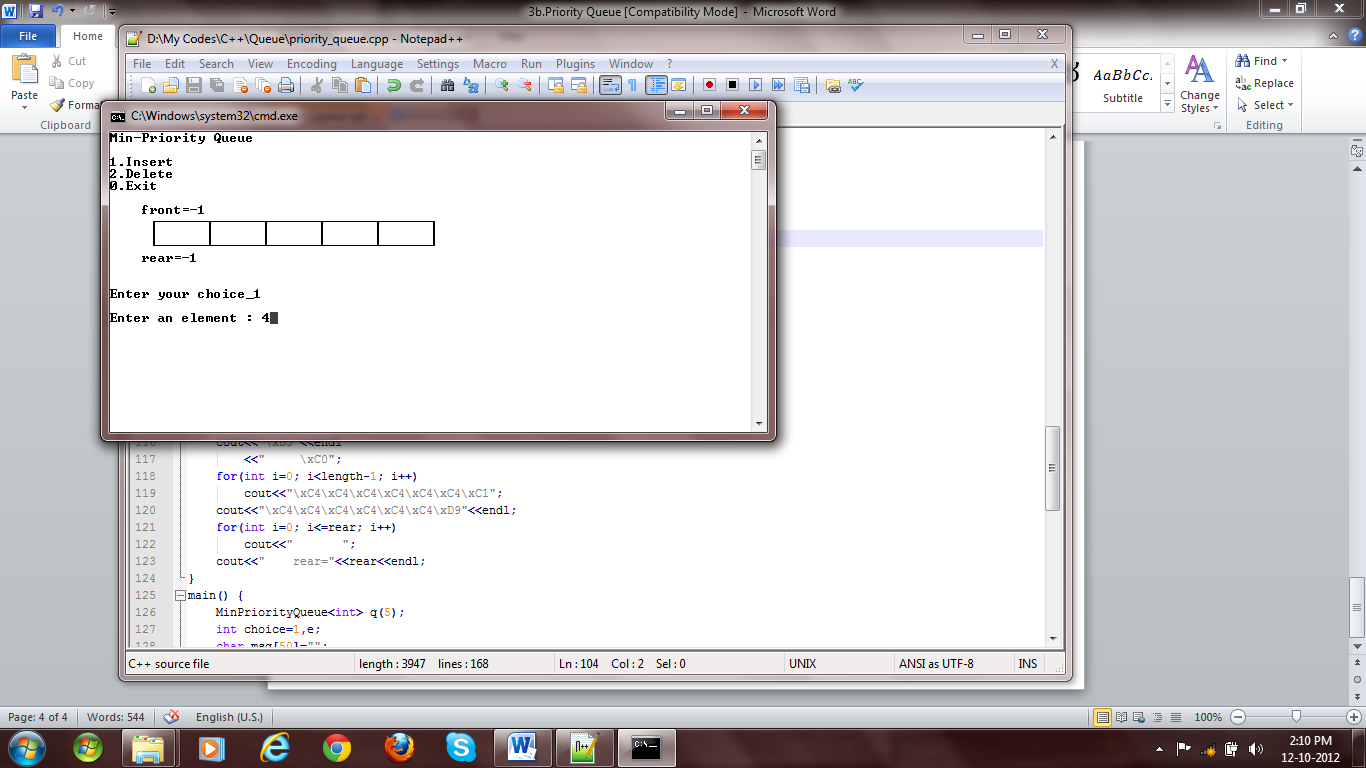


Fig 1: Empty Queue

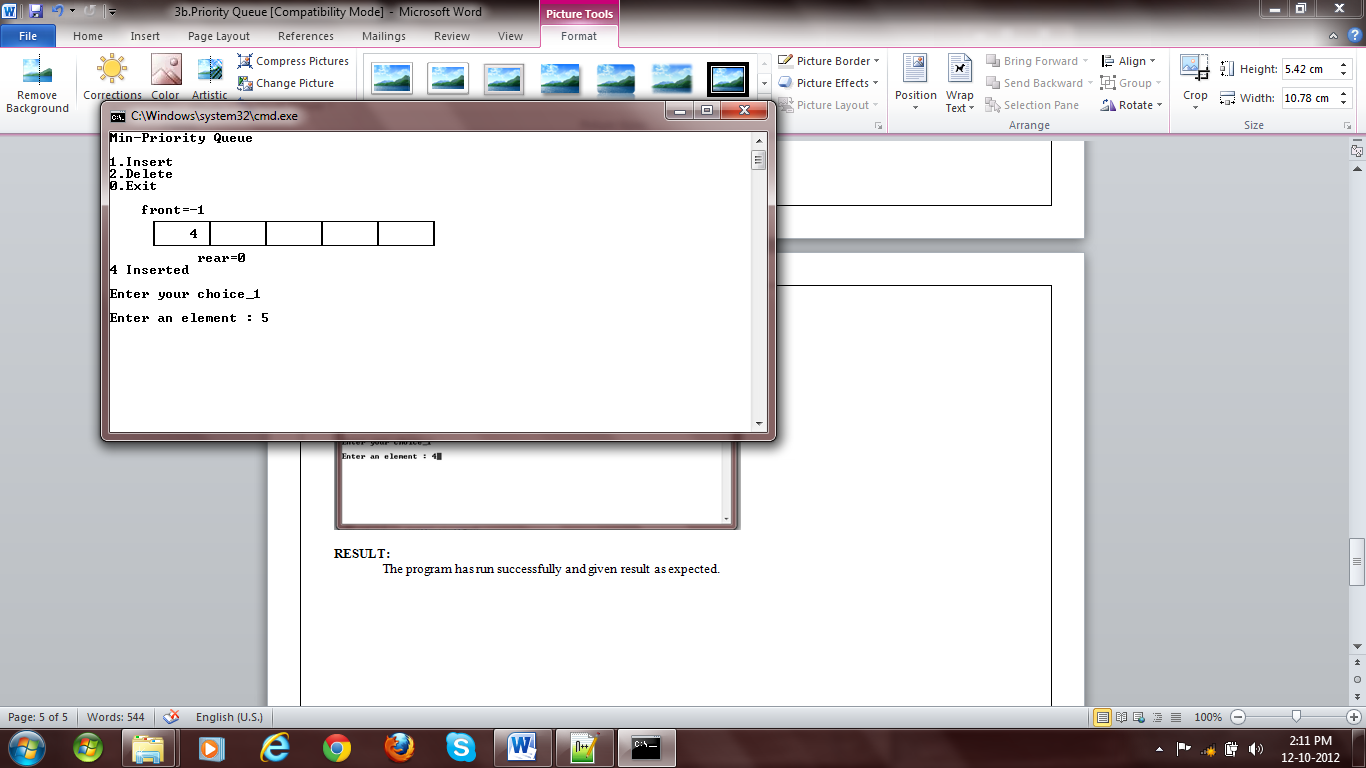


Fig 2: Insertion

# \* Author : Rakesh Malik

# \* Date : 24/07/2012

# \* Subject : Generic MinPriorityQueue Class

# \* Assignment no. :

# \*/

# #include <iostream>

# #include <cstdio>

# #include <cstdlib>

# #include "..\myexception.h"

# #include <iomanip>

# using namespace std;

# using namespace exception;

# template <class DT> void bubbleSort(DT\* arr, int n) {

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

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

# if(arr[j]>arr[j+1]) {

# DT temp=arr[j];

# arr[j]=arr[j+1];

# arr[j+1]=temp;

# }

# }

# template <class DT> class MinPriorityQueue {

# DT \*buf;

# int rear;

# int length;

# public:

# MinPriorityQueue(const MinPriorityQueue<DT>&);

# MinPriorityQueue(int);

# ~MinPriorityQueue();

# bool isFull();

# bool isEmpty();

# void enqueue(DT);

# DT dequeue();

# int getLength();

# int getNumberOfElements();

# void display();

# void clear();

# };

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

# MinPriorityQueue::length=q.length;

# MinPriorityQueue::front=q.front;

# MinPriorityQueue::rear=q.rear;

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

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

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

# }

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

# if(length>=0)

# {

# MinPriorityQueue::length=length;

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

# }

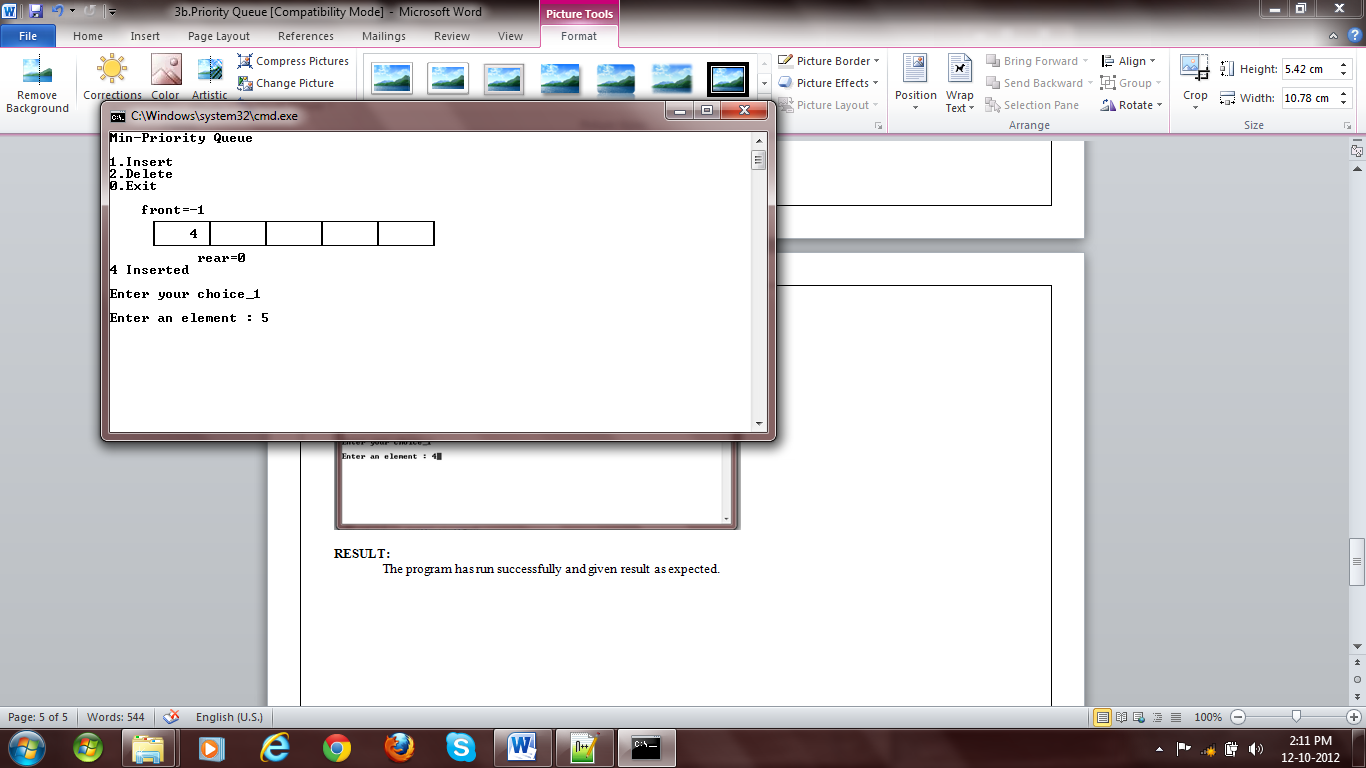


Fig 3: Insertion

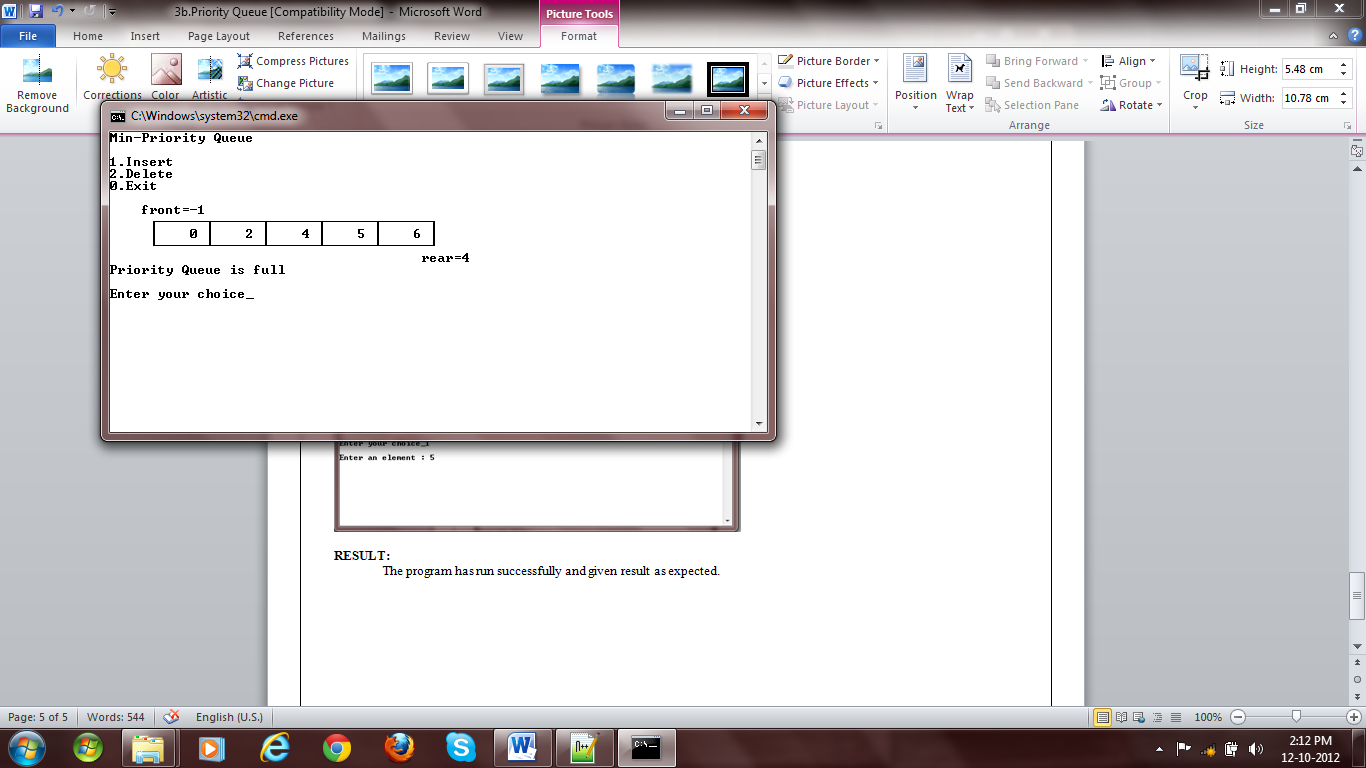


Fig 4: Queue full message while insertion

# else

# {

# MinPriorityQueue::length=0;

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

# }

# MinPriorityQueue::rear=-1;

# }

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

# delete MinPriorityQueue::buf;

# }

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

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

# }

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

# return MinPriorityQueue::rear==-1;

# }

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

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

# MinPriorityQueue::rear++;

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

# cout<<rear<<endl;

# bubbleSort(buf, rear+1);

# }

# else

# throw QueueFullException();

# }

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

# if(!MinPriorityQueue::isEmpty())

# {

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

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

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

# MinPriorityQueue::rear--;

# return e;

# }

# else

# throw QueueEmptyException();

# }

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

# return MinPriorityQueue::length;

# }

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

# return MinPriorityQueue::rear+1;

# }

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

# MinPriorityQueue::rear=-1;

# }

# template <class DT> void MinPriorityQueue<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++)

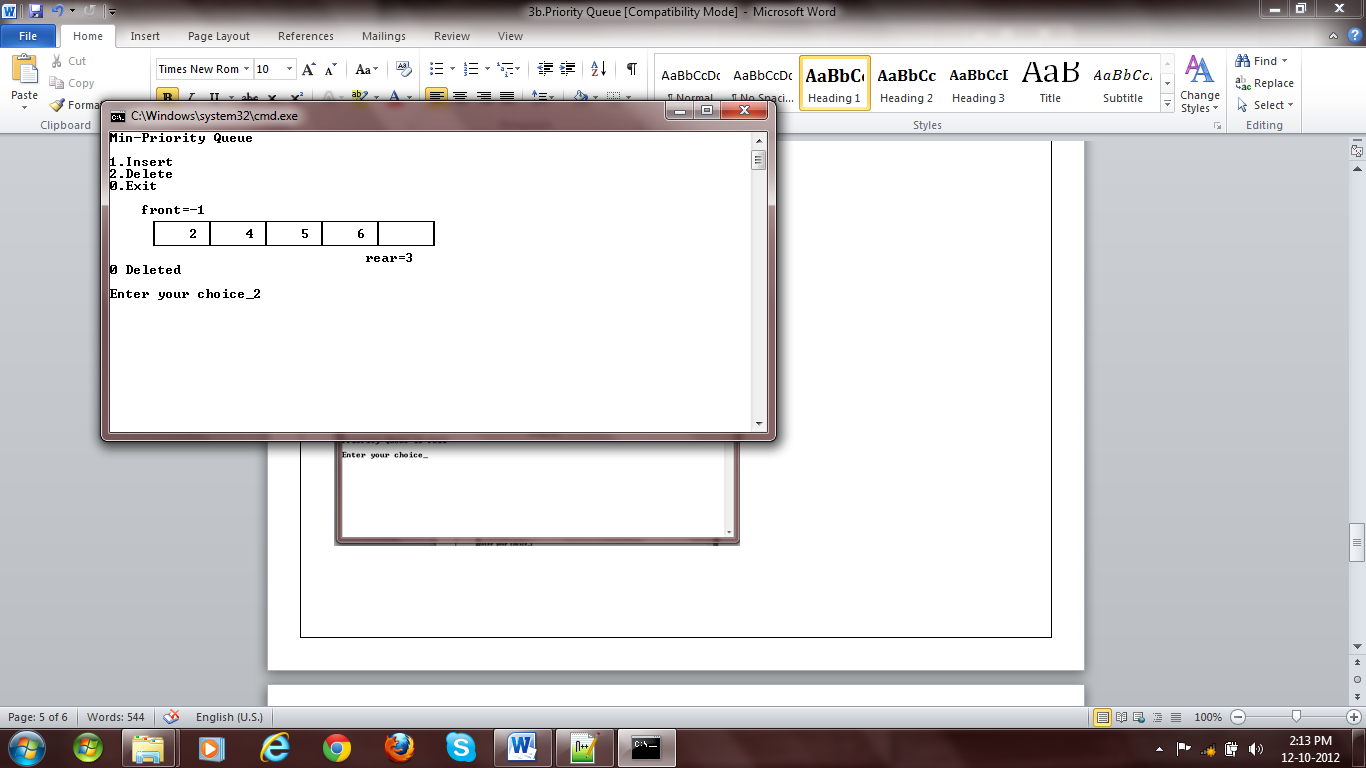


Fig 5: Deletion

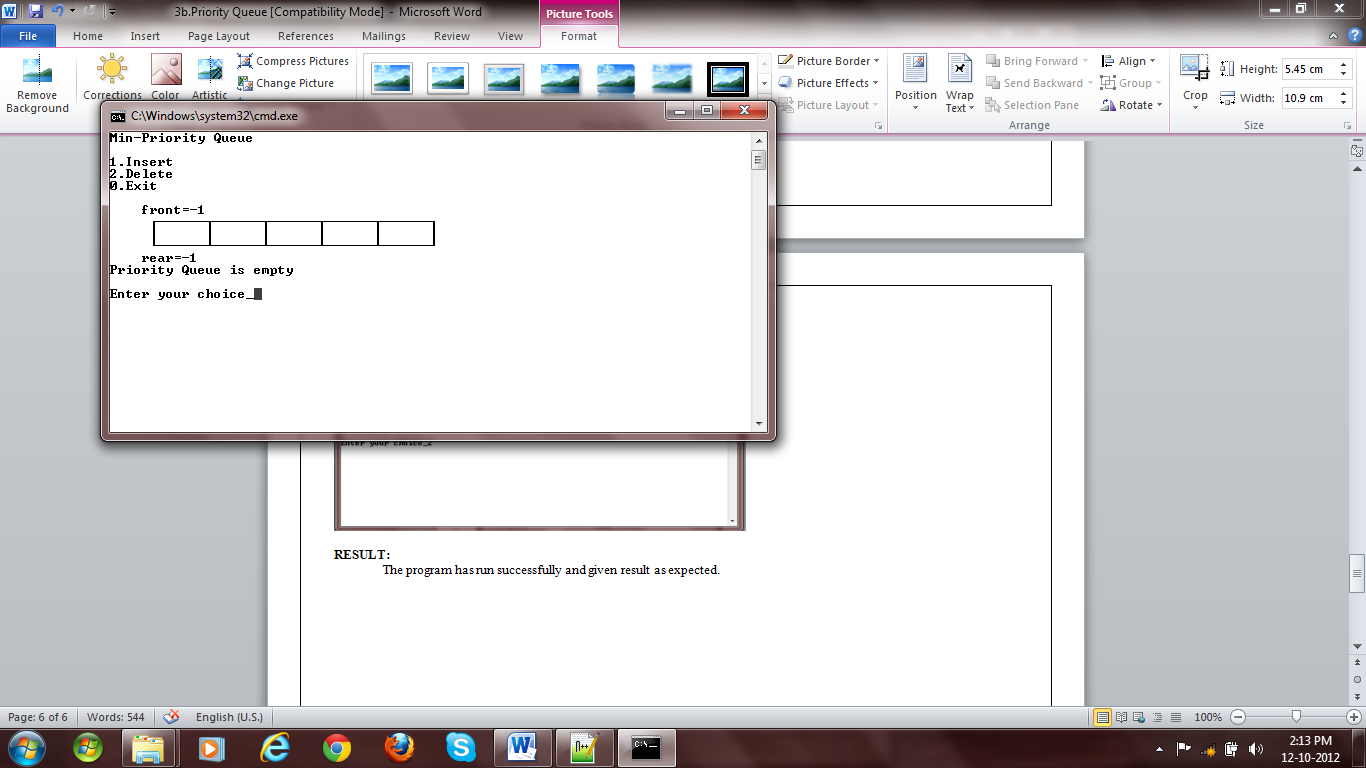


Fig 6: Queue empty message while deleting

# 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() {

# MinPriorityQueue<int> q(5);

# int choice=1,e;

# char msg[50]="";

# while(choice) {

# system("cls");

# cout<<"Min-Priority 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, "Priority 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, "Priority Queue is empty");

# }

# break;

# default:

# sprintf(msg, "Wrong choice");

# }

# }

# }

# RESULT:

The program has run successfully and given result as expected.