**1012: 顺序栈**

Time Limit: 100 Sec  Memory Limit: 128 MB  
Submit: 67  Solved: 56  
[[Submit](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/submitpage.php?id=1012)][[Status](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/problemstatus.php?id=1012)][[Web Board](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/bbs.php?pid=1012)]

**Description**

**已知顺序栈类的定义、构造函数及主函数的代码如下，请完成类的其它成员函数，完成相应的输出。**

const int StackSize=5; //顺序栈的最大长度（请勿改动）

template <class T>       //定义模板类SeqStack

class SeqStack

{

public:

    SeqStack( ) ;            //构造函数，栈的初始化

       ~SeqStack( );            //析构函数

    void Push(T x);          //将元素x入栈

    T Pop( );                //将栈顶元素弹出

    T GetTop( );            //取栈顶元素（并不删除）

       bool Empty( );           //判断栈是否为空

private:

    T data[StackSize];      //存放栈元素的数组

    int top;                //栈顶指针，指示栈顶元素在数组中的下标

};

/\*

 \* 前置条件：栈不存在

 \* 输    入：无

 \* 功    能：栈的初始化

 \* 输    出：无

 \* 后置条件：构造一个空栈

 \*/

template <class T>

SeqStack<T>::SeqStack( )

{

       top=-1;

}

 /\*

 \* 前置条件：栈已存在

 \* 输    入：无

 \* 功    能：销毁栈

 \* 输    出：无

 \* 后置条件：释放栈所占用的存储空间

 \*/

template <class T>

SeqStack<T>::~SeqStack( )

{

}

int main()

{

       SeqStack<int> s;

       int x;

       while(1)

       {

              cin>>x;

              if(!x) break;

              try{

                     s.Push(x);

              }

              catch(const char \*ms){

                     cout<<"Push:"<<ms<<endl;

              }

       }

       cout<<"Gettop:"<<s.GetTop()<<endl;

while(!s.Empty())

       {

              cout<<s.Pop()<<" ";

       }

       cout<<endl;

       try{

              cout<<"Gettop:"<<s.GetTop()<<endl;

       }

       catch(const char \*ms){

                     cout<<"Gettop:"<<ms<<endl;

       }

       return 0;

}

**Input**

**Output**

**Sample Input**

1 2 3 4 5 6 7 0

**Sample Output**

Push:Overflow

Push:Overflow

Gettop:5

5 4 3 2 1

Gettop:Downflow

#include<iostream>

using namespace std;

const int StackSize=5; //顺序栈的最大长度（请勿改动）

template <class T> //定义模板类SeqStack

class SeqStack

{

public:

SeqStack( ) ; //构造函数，栈的初始化

~SeqStack( ); //析构函数

void Push(T x); //将元素x入栈

T Pop( ); //将栈顶元素弹出

T GetTop( ); //取栈顶元素（并不删除）

bool Empty( ); //判断栈是否为空

private:

T data[StackSize]; //存放栈元素的数组

int top; //栈顶指针，指示栈顶元素在数组中的下标

};

/\*

\* 前置条件：栈不存在

\* 输 入：无

\* 功 能：栈的初始化

\* 输 出：无

\* 后置条件：构造一个空栈

\*/

template <class T>

SeqStack<T>::SeqStack( )

{

top=-1;

}

template <class T>

void SeqStack<T>::Push(T x)

{

if(top==4) throw "Overflow";

top++;

data[top]=x;

}

template <class T>

T SeqStack<T>::Pop( )

{

int p;

p=top;

top--;

return data[p];

}

template <class T>

T SeqStack<T>::GetTop( )

{

if(top==-1) throw "Downflow";

return data[top];

}

template <class T>

bool SeqStack<T>::Empty( )

{

if(top==-1) return true;

else return false;

}

/\*

\* 前置条件：栈已存在

\* 输 入：无

\* 功 能：销毁栈

\* 输 出：无

\* 后置条件：释放栈所占用的存储空间

\*/

template <class T>

SeqStack<T>::~SeqStack( )

{

}

int main()

{

SeqStack<int> s;

int x;

while(1)

{

cin>>x;

if(!x) break;

try{

s.Push(x);

}

catch(const char \*ms){

cout<<"Push:"<<ms<<endl;

}

}

cout<<"Gettop:"<<s.GetTop()<<endl;

while(!s.Empty())

{

cout<<s.Pop()<<" ";

}

cout<<endl;

try{

cout<<"Gettop:"<<s.GetTop()<<endl;

}

catch(const char \*ms){

cout<<"Gettop:"<<ms<<endl;

}

return 0;

}

**1013: 链式栈**

Time Limit: 1 Sec  Memory Limit: 128 MB  
Submit: 51  Solved: 49  
[[Submit](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/submitpage.php?id=1013)][[Status](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/problemstatus.php?id=1013)][[Web Board](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/bbs.php?pid=1013)]

**Description**

**已知链栈类的定义、构造函数及main函数如下，请完成其余的成员函数实现。**

template <class T>

class LinkStack

{

public:

    LinkStack( );              //构造函数，置空链栈

    ~LinkStack( );             //析构函数，释放链栈中各结点的存储空间

    void Push(T x);           //将元素x入栈

    T Pop( );                  //将栈顶元素出栈

    T GetTop( );               //取栈顶元素（并不删除）

    bool Empty( );             //判断链栈是否为空栈

private:

    Node<T> \*top;             //栈顶指针即链栈的头指针

};

/\*

 \* 前置条件：栈不存在

 \* 输    入：无

 \* 功    能：栈的初始化

 \* 输    出：无

 \* 后置条件：构造一个空栈

 \*/

template <class T>

LinkStack<T>::LinkStack( )

{

       top=NULL;

}

 int main()

{

       LinkStack<char> s;

       char ch;

       while(1)

       {

              cin>>ch;

              if(ch=='#') break;

              s.Push(ch);

       }

       cout<<"Gettop:"<<s.GetTop()<<endl;

       while(!s.Empty())

       {

              cout<<s.Pop()<<" ";

       }

       cout<<endl;

       try{

              cout<<"Gettop:"<<s.GetTop()<<endl;

       }

       catch(const char \*ms){

                     cout<<"Gettop:"<<ms<<endl;

       }

       return 0;

}

**Input**

**Output**

**Sample Input**

asdfgh#

**Sample Output**

Gettop:h

h g f d s a

Gettop:Downflow

#include<iostream>

using namespace std;

template <class T>

struct Node

{

T data;

Node<T> \*next;

};

template <class T>

class LinkStack

{

public:

LinkStack( ); //构造函数，置空链栈

~LinkStack( ); //析构函数，释放链栈中各结点的存储空间

void Push(T x); //将元素x入栈

T Pop( ); //将栈顶元素出栈

T GetTop( ); //取栈顶元素（并不删除）

bool Empty( ); //判断链栈是否为空栈

private:

Node<T> \*top; //栈顶指针即链栈的头指针

};

/\*

\* 前置条件：栈不存在

\* 输 入：无

\* 功 能：栈的初始化

\* 输 出：无

\* 后置条件：构造一个空栈

\*/

template <class T>

LinkStack<T>::LinkStack( )

{

top=NULL;

}

template <class T>

LinkStack<T>::~LinkStack( )

{

Node<T> \*p;

while(1)

{

p=top;

if(!p) break;

top=top->next;

delete p;

}

}

template <class T>

void LinkStack<T>::Push(T x)

{

Node<T> \*s;

s=new Node<T>;

s->data=x;

s->next=top;

top=s;

}

template <class T>

T LinkStack<T>::Pop( )

{

T x;

Node<T> \*s;

s=top;

x=top->data;

top=top->next;

delete s;

return x;

}

template <class T>

T LinkStack<T>::GetTop( )

{

if(!top) throw "Downflow";

return top->data;

}

template <class T>

bool LinkStack<T>::Empty()

{

if(top) return false;

else return true;

}

int main()

{

LinkStack<char> s;

char ch;

while(1)

{

cin>>ch;

if(ch=='#') break;

s.Push(ch);

}

cout<<"Gettop:"<<s.GetTop()<<endl;

while(!s.Empty())

{

cout<<s.Pop()<<" ";

}

cout<<endl;

try{

cout<<"Gettop:"<<s.GetTop()<<endl;

}

catch(const char \*ms){

cout<<"Gettop:"<<ms<<endl;

}

return 0;

}

**1014: 链式队列**

Time Limit: 1 Sec  Memory Limit: 128 MB  
Submit: 69  Solved: 54  
[[Submit](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/submitpage.php?id=1014)][[Status](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/problemstatus.php?id=1014)][[Web Board](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/bbs.php?pid=1014)]

**Description**

**已知链式队列类的定义、部分实现及主函数代码（勿改动）如下：**

template <class T>  
struct Node  
{  
      T data;  
      Node<T> \*next;

};

template <class T>  
class LinkQueue  
{  
public:  
    LinkQueue( );      //构造函数，初始化一个仅有头结点的空队列  
    ~LinkQueue( );      //析构函数，释放链队列中各结点的存储空间  
    void EnQueue(T x);  //将元素x入队  
    T DeQueue( );       //将队头元素出队，若队列为空，抛出异常“Downflow”，否则函数返回值为队头元素值  
    T GetQueue( );     //取链队列的队头元素，若队列为空，抛出异常“Downflow”，否则函数返回值为队头元素值  
    bool Empty( );     //判断链队列是否为空，为空返回true，否则返回false  
private:  
    Node<T> \*front, \*rear;  //队头和队尾指针，分别指向头结点和终端结点  
};

/\*  
 \* 前置条件：队列不存在  
 \* 输    入：无  
 \* 功    能：初始化队列  
 \* 输    出：无  
 \* 后置条件：创建一个空队列  
 \*/

template <class T>  
LinkQueue<T>::LinkQueue( )  
{  
 front=rear=new Node<T>;   
}

/\*  
 \* 前置条件：队列存在  
 \* 输    入：无  
 \* 功    能：销毁队列  
 \* 输    出：无  
 \* 后置条件：释放队列所占用的存储空间  
 \*/

template <class T>  
LinkQueue<T>::~LinkQueue( )  
{  
 while(front)  
 {  
  Node <T> \*p;  
  p=front->next;  
        delete front;  
     front=p;  
 }  
}

/\*  
 \* 前置条件：队列已存在  
 \* 输    入：无  
 \* 功    能：判断队列是否为空  
 \* 输    出：如果队列为空，返回true，否则，返回false  
 \* 后置条件：队列不变  
 \*/

template <class T>   
bool LinkQueue<T>::Empty( )  
{  
    return front==rear;  
}

int main()  
{  
 LinkQueue <int> Q1;  
 int x;  
 while(1)  
 {  
  cin>>x;  
  if(!x) break;  
  Q1.EnQueue(x);  
 }  
 cout<<"DeQueue:";  
 while(!Q1.Empty())  
 {  
  x=Q1.DeQueue();  
  cout<<x<<" ";  
 }  
 try{  
  x=Q1.DeQueue();  
  cout<<x<<" ";  
 }  
 catch(const char \*ms)  
 {  
  cout<<ms<<endl;  
 }  
 cout<<"GetQueue:";  
 try{  
  x=Q1.GetQueue();  
  cout<<x<<" ";  
 }  
 catch(const char \*ms)  
 {  
  cout<<ms<<endl;  
 }  
 return 0;  
}

请实现其它的未实现的成员函数，达到任务要求。

**Input**

**Output**

**Sample Input**

1 2 3 4 5 0

**Sample Output**

DeQueue:1 2 3 4 5 Downflow

GetQueue:Downflow

#include<iostream>

using namespace std;

template <class T>

struct Node

{

T data;

Node<T> \*next;

};

template <class T>

class LinkQueue

{

public:

LinkQueue( ); //构造函数，初始化一个仅有头结点的空队列

~LinkQueue( ); //析构函数，释放链队列中各结点的存储空间

void EnQueue(T x); //将元素x入队

T DeQueue( ); //将队头元素出队，若队列为空，抛出异常"Downflow"，否则函数返回值为队头元素值

T GetQueue( ); //取链队列的队头元素，若队列为空，抛出异常"Downflow"，否则函数返回值为队头元素值

bool Empty( ); //判断链队列是否为空，为空返回true，否则返回false

private:

Node<T> \*front, \*rear; //队头和队尾指针，分别指向头结点和终端结点

};

template <class T>

LinkQueue<T>::LinkQueue( )

{

front=rear=new Node<T>;

}

/\*

\* 前置条件：队列存在

\* 输 入：无

\* 功 能：销毁队列

\* 输 出：无

\* 后置条件：释放队列所占用的存储空间

\*/

template <class T>

LinkQueue<T>::~LinkQueue( )

{

while(front)

{

Node <T> \*p;

p=front->next;

delete front;

front=p;

}

}

/\*

\* 前置条件：队列已存在

\* 输 入：无

\* 功 能：判断队列是否为空

\* 输 出：如果队列为空，返回true，否则，返回false

\* 后置条件：队列不变

\*/

template <class T>

bool LinkQueue<T>::Empty( )

{

return front==rear;

}

template <class T>

void LinkQueue<T>::EnQueue(T x)

{

Node<T> \*s;

s=new Node<T>;

s->data=x;

s->next=NULL;

rear->next=s;

rear=s;

}

template <class T>

T LinkQueue<T>::DeQueue( )

{

Node<T> \*p;

T x;

if(front==rear) throw "Downflow";

p=front->next;

x=p->data;

front->next=p->next;

if(p->next==NULL) rear=front;

delete p;

return x;

}

template <class T>

T LinkQueue<T>::GetQueue( )

{

if(front==rear)

throw "Downflow";

return front->next->data;

}

int main()

{

LinkQueue <int> Q1;

int x;

while(1)

{

cin>>x;

if(!x) break;

Q1.EnQueue(x);

}

cout<<"DeQueue:";

while(!Q1.Empty())

{

x=Q1.DeQueue();

cout<<x<<" ";

}

try{

x=Q1.DeQueue();

cout<<x<<" ";

}

catch(const char \*ms)

{

cout<<ms<<endl;

}

cout<<"GetQueue:";

try{

x=Q1.GetQueue();

cout<<x<<" ";

}

catch(const char \*ms)

{

cout<<ms<<endl;

}

return 0;

}

**1015: 循环队列（1）**

Time Limit: 1 Sec  Memory Limit: 128 MB  
Submit: 48  Solved: 43  
[[Submit](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/submitpage.php?id=1015)][[Status](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/problemstatus.php?id=1015)][[Web Board](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/bbs.php?pid=1015)]

**Description**

**循环队列中利用头尾指针front、rear值得关系实现队满（仅剩一个空闲单元时即视为队满）、队空条件判断。循环队列类的定义、部分实现及主函数代码如下：**

#include <iostream>  
#include <string>  
using namespace std;  
const int QueueSize=5;  
template <class T>        //定义模板类CirQueue  
class CirQueue  
{  
public:  
    CirQueue( );                 //构造函数，置空队  
    ~ CirQueue( );               //析构函数  
    void EnQueue(T x);           //将元素x入队  
    T DeQueue( );                //将队头元素出队  
    T GetQueue( );               //取队头元素（并不删除）  
    bool Empty( );               //判断队列是否为空，空返回true，否则返回false  
 bool Full();                 //判断队列是否为满，满返回true，否则返回false  
private:  
    T data[QueueSize];           //存放队列元素的数组  
    int front, rear;    //队头和队尾指针，分别指向队头元素所在数组的前一下标和队尾元素的数组下标  
};

/\*  
 \* 前置条件：队列不存在  
 \* 输    入：无  
 \* 功    能：初始化队列  
 \* 输    出：无  
 \* 后置条件：创建一个空队列  
 \*/

template <class T>  
CirQueue<T>::CirQueue( )   
{  
 front=rear=QueueSize-1;  
}

/\*  
 \* 前置条件：队列已存在  
 \* 输    入：无  
 \* 功    能：销毁队列  
 \* 输    出：无  
 \* 后置条件：释放队列所占用的存储空间  
 \*/

template <class T>  
CirQueue<T>::~CirQueue( )  
{

}

/\*  
 \* 前置条件：队列已存在  
 \* 输    入：无  
 \* 功    能：判断队列是否为空  
 \* 输    出：如果队列为空，返回1，否则，返回0  
 \* 后置条件：队列不变  
 \*/

template <class T>   
bool CirQueue<T>::Empty( )   
{  
    return front==rear;  
}  
/\*  
 \* 前置条件：队列已存在  
 \* 输    入：无  
 \* 功    能：判断队列是否为满  
 \* 输    出：如果队列为满，返回1，否则，返回0  
 \* 后置条件：队列不变  
 \*/

template <class T>   
bool CirQueue<T>::Full( )   
{  
    return (rear+1) % QueueSize ==front;   
}  
int main()  
{  
 CirQueue<string> Q;  
 string x;  
 while(1){  
  cin>>x;  
        if(x=="#") break;  
  try{  
   cout<<"EnQueue:";  
   Q.EnQueue(x);  
   cout<<x<<"\n";  
  }  
  catch(const char \*ms)  
  {  
   cout<<x<<" "<<ms<<endl;  
  }

 }  
 while(!Q.Empty())  
 {  
  x=Q.DeQueue();  
  cout<<"DeQueue:"<<x<<endl;  
 }  
 try{  
  x=Q.GetQueue();  
 }  
 catch(const char \*ms)  
 {  
  cout<<"GetQueue:The queue is empty,"<<ms<<endl;  
 }  
 return 0;  
}

请实现其它未实现的成员函数，达到任务要求。

**Input**

**Output**

**Sample Input**

zhang sun li zhao wang xia #

**Sample Output**

EnQueue:zhang

EnQueue:sun

EnQueue:li

EnQueue:zhao

EnQueue:wang Overflow

EnQueue:xia Overflow

DeQueue:zhang

DeQueue:sun

DeQueue:li

DeQueue:zhao

GetQueue:The queue is empty,Downflow

#include <iostream>

#include <string>

using namespace std;

const int QueueSize=5;

template <class T> //定义模板类CirQueue

class CirQueue

{

public:

CirQueue( ); //构造函数，置空队

~ CirQueue( ); //析构函数

void EnQueue(T x); //将元素x入队

T DeQueue( ); //将队头元素出队

T GetQueue( ); //取队头元素（并不删除）

bool Empty( ); //判断队列是否为空，空返回true，否则返回false

bool Full(); //判断队列是否为满，满返回true，否则返回false

private:

T data[QueueSize]; //存放队列元素的数组

int front, rear; //队头和队尾指针，分别指向队头元素所在数组的前一下标和队尾元素的数组下标

};

/\*

\* 前置条件：队列不存在

\* 输 入：无

\* 功 能：初始化队列

\* 输 出：无

\* 后置条件：创建一个空队列

\*/

template <class T>

CirQueue<T>::CirQueue( )

{

front=rear=QueueSize-1;

}

/\*

\* 前置条件：队列已存在

\* 输 入：无

\* 功 能：销毁队列

\* 输 出：无

\* 后置条件：释放队列所占用的存储空间

\*/

template <class T>

CirQueue<T>::~CirQueue( )

{

}

/\*

\* 前置条件：队列已存在

\* 输 入：无

\* 功 能：判断队列是否为空

\* 输 出：如果队列为空，返回1，否则，返回0

\* 后置条件：队列不变

\*/

template <class T>

bool CirQueue<T>::Empty( )

{

return front==rear;

}

/\*

\* 前置条件：队列已存在

\* 输 入：无

\* 功 能：判断队列是否为满

\* 输 出：如果队列为满，返回1，否则，返回0

\* 后置条件：队列不变

\*/

template <class T>

bool CirQueue<T>::Full( )

{

return (rear+1) % QueueSize ==front;

}

template <class T>

void CirQueue<T>::EnQueue(T x)

{

if((rear+1) % QueueSize ==front) throw "Overflow";

if(rear==4) rear=0;

else rear++;

data[rear]=x;

}

template <class T>

T CirQueue<T>::DeQueue()

{

T x;

int f;

if(rear==front) throw "Downflow";

if(front==QueueSize-1)

{

f=0;

front=0;

}

else {f=front+1;front++;}

x=data[f];

return x;

}

template <class T>

T CirQueue<T>::GetQueue( )

{

if(rear==front) throw "Downflow";

return data[front];

}

int main()

{

CirQueue<string> Q;

string x;

while(1){

cin>>x;

if(x=="#") break;

try{

cout<<"EnQueue:";

Q.EnQueue(x);

cout<<x<<"\n";

}

catch(const char \*ms)

{

cout<<x<<" "<<ms<<endl;

}

}

while(!Q.Empty())

{

x=Q.DeQueue();

cout<<"DeQueue:"<<x<<endl;

}

try{

x=Q.GetQueue();

}

catch(const char \*ms)

{

cout<<"GetQueue:The queue is empty,"<<ms<<endl;

}

return 0;

}

**1016: 循环队列（2）**

Time Limit: 1 Sec  Memory Limit: 128 MB  
Submit: 45  Solved: 38  
[[Submit](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/submitpage.php?id=1016)][[Status](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/problemstatus.php?id=1016)][[Web Board](http://221.203.21.203:8001/rwt/USTL/http/GV6T6N3UFZ5UGLSSGM3B/OnlineJudge/bbs.php?pid=1016)]

**Description**

**在循环队列中增加一个count变量，用来记录当前队列中数据个数，利用该值为0或QueueSize来判断队空或队满条件判断。循环队列类的定义、部分实现及主函数代码如下：**

#include <iostream>  
#include <string>  
using namespace std;  
const int QueueSize=5;  
template <class T>        //定义模板类CirQueue  
class CirQueue  
{  
public:  
    CirQueue( );                 //构造函数，置空队  
    ~CirQueue( );                //析构函数  
    void EnQueue(T x);           //将元素x入队  
    T DeQueue( );                //将队头元素出队  
    T GetQueue( );               //取队头元素（并不删除）  
    bool Empty( );               //判断队列是否为空，空返回true，否则返回false  
 bool Full();                 //判断队列是否为满，满返回true，否则返回false  
private:  
    T data[QueueSize];           //存放队列元素的数组  
    int front, rear;    //队头和队尾指针，分别指向队头元素所在数组的前一下标和队尾元素的数组下标  
    int count;          //记录队列中数据个数  
};

/\*  
 \* 前置条件：队列不存在  
 \* 输    入：无  
 \* 功    能：初始化队列  
 \* 输    出：无  
 \* 后置条件：创建一个空队列  
 \*/

template <class T>  
CirQueue<T>::CirQueue( )   
{  
 front=rear=QueueSize-1;  
 count=0;  
}

/\*  
 \* 前置条件：队列已存在  
 \* 输    入：无  
 \* 功    能：销毁队列  
 \* 输    出：无  
 \* 后置条件：释放队列所占用的存储空间  
 \*/

template <class T>  
CirQueue<T>::~CirQueue( )  
{

}

/\*  
 \* 前置条件：队列已存在  
 \* 输    入：无  
 \* 功    能：删除队头元素  
 \* 输    出：如果删除成功，返回被删元素值，否则，抛出删除异常  
 \* 后置条件：如果删除成功，队头减少了一个元素  
 \*/

template <class T>   
T CirQueue<T>::DeQueue( )  
{  
 if(Empty()) throw "Downflow";   
    count--;  
 front=(front+1) % QueueSize;    //队头指针在循环意义下加1  
    return data[front];             //读取并返回出队前的队头元素，注意队头指针  
 }  
                                    
/\*  
 \* 前置条件：队列已存在  
 \* 输    入：无  
 \* 功    能：读取队头元素  
 \* 输    出：若队列不空，返回队头元素  
 \* 后置条件：队列不变  
 \*/

template <class T>  
T CirQueue<T>::GetQueue( )  
{     
    int i;  
    if (Empty()) throw "Downflow";   
    i=(front+1) % QueueSize;  //注意不要给队头指针赋值  
    return data[i];  
}

int main()  
{  
 CirQueue<string> Q;  
 string x;  
 while(1){  
  cin>>x;  
        if(x=="#") break;  
  try{  
   cout<<"EnQueue:";  
   Q.EnQueue(x);  
   cout<<x<<"\n";  
  }  
  catch(const char \*ms)  
  {  
   cout<<x<<" "<<ms<<endl;  
  }  
 }  
 while(!Q.Empty())  
 {  
  x=Q.DeQueue();  
  cout<<"DeQueue:"<<x<<endl;  
 }  
 try{  
  x=Q.GetQueue();  
 }  
 catch(const char \*ms)  
 {  
  cout<<"GetQueue:The queue is empty,"<<ms<<endl;  
 }  
 return 0;  
}

请实现其它未实现的成员函数，达到任务要求。

**Input**

**Output**

**Sample Input**

zhang sun li zhao wang xia #

**Sample Output**

EnQueue:zhang

EnQueue:sun

EnQueue:li

EnQueue:zhao

EnQueue:wang

EnQueue:xia Overflow

DeQueue:zhang

DeQueue:sun

DeQueue:li

DeQueue:zhao

DeQueue:wang

GetQueue:The queue is empty,Downflow

#include <iostream>

#include <string>

using namespace std;

const int QueueSize=5;

template <class T> //定义模板类CirQueue

class CirQueue

{

public:

CirQueue( ); //构造函数，置空队

~CirQueue( ); //析构函数

void EnQueue(T x); //将元素x入队

T DeQueue( ); //将队头元素出队

T GetQueue( ); //取队头元素（并不删除）

bool Empty( ); //判断队列是否为空，空返回true，否则返回false

bool Full(); //判断队列是否为满，满返回true，否则返回false

private:

T data[QueueSize]; //存放队列元素的数组

int front, rear; //队头和队尾指针，分别指向队头元素所在数组的前一下标和队尾元素的数组下标

int count; //记录队列中数据个数

};

/\*

\* 前置条件：队列不存在

\* 输 入：无

\* 功 能：初始化队列

\* 输 出：无

\* 后置条件：创建一个空队列

\*/

template <class T>

CirQueue<T>::CirQueue( )

{

front=rear=QueueSize-1;

count=0;

}

/\*

\* 前置条件：队列已存在

\* 输 入：无

\* 功 能：销毁队列

\* 输 出：无

\* 后置条件：释放队列所占用的存储空间

\*/

template <class T>

CirQueue<T>::~CirQueue( )

{

}

/\*

\* 前置条件：队列已存在

\* 输 入：无

\* 功 能：删除队头元素

\* 输 出：如果删除成功，返回被删元素值，否则，抛出删除异常

\* 后置条件：如果删除成功，队头减少了一个元素

\*/

template <class T>

T CirQueue<T>::DeQueue( )

{

if(Empty()) throw "Downflow";

count--;

T x;

x=data[front];

front=(front+1) % QueueSize; //队头指针在循环意义下加1

return x; //读取并返回出队前的队头元素，注意队头指针

}

/\*

\* 前置条件：队列已存在

\* 输 入：无

\* 功 能：读取队头元素

\* 输 出：若队列不空，返回队头元素

\* 后置条件：队列不变

\*/

template <class T>

T CirQueue<T>::GetQueue( )

{

int i;

if (Empty()) throw "Downflow";

i=(front+1) % QueueSize; //注意不要给队头指针赋值

return data[i];

}

template <class T>

void CirQueue<T>::EnQueue(T x)

{

if(Full()) throw "Overflow";

data[rear]=x;

rear=(rear+1)%QueueSize;

count++;

}

template <class T>

bool CirQueue<T>::Empty()

{

return count==0;

}

template <class T>

bool CirQueue<T>::Full()

{

return count==QueueSize;

}

int main()

{

CirQueue<string> Q;

string x;

while(1){

cin>>x;

if(x=="#") break;

try{

cout<<"EnQueue:";

Q.EnQueue(x);

cout<<x<<"\n";

}

catch(const char \*ms)

{

cout<<x<<" "<<ms<<endl;

}

}

while(!Q.Empty())

{

x=Q.DeQueue();

cout<<"DeQueue:"<<x<<endl;

}

try{

x=Q.GetQueue();

}

catch(const char \*ms)

{

cout<<"GetQueue:The queue is empty,"<<ms<<endl;

}

return 0;

}