1.尝试用代码实现普通队列，并完成如下操作

（1）输入1-10的数字组成普通线性队列；

（2）判断此时队列的队首元素；

（3）打印出队列操作后的队首元素；

（4）由于普通线性队列的限制，此时尝试使用循环队列来解决上述问题。

参考答案：

线性队列初始化：



入队操作



出队及获取队首元素的操作：



循环队列的实现：

#include <iostream>

using namespace std;

class Queue{

private:

int \*data; //定义指向整型的指针，从而动态开辟内存

int head,tail,length,count; //head指向队首,tail指向队尾,length表示队列的长度,count用于记录队列中元素个数,从而判断队列是否已满

public:

Queue (int length\_input){ //构造函数,对新声明的队列对象进行初始化

data = new int [length\_input]; //动态开辟100个整型数据的空间

length = length\_input; //为队列的长度赋初始值

head = 0; //起初head为0，表示head值未改变的情况下,一直指向首元素

tail = -1; //tail初始值为-1，表示此时队列为空,无任何数据元素

count = 0;

}

~Queue(){

delete [] data; //析构函数,删除动态开辟的内存

}

void push(int element){ //入队操作,只能从队列的尾部插入数据元素

if(count < length){ //队列未满的时候才能插入,否则则插入失败

tail = (tail + 1) % length; //分两种情况,如果队尾指针此时并未指向队列的最后一位,那么队尾指针直接前移,而当队尾指针此时指向最后一位时

data[tail] = element; //那么当队列未满时,则队尾指针将跳转至数组起始位置，再将数据元素插入队尾指针指向的位置

++count; //入队成功,队列中元素数量加一

}

}

void pop(){

if(count < 0){ //队列为空,出队失败

return;

}

head = (head + 1) % length; //同样,根据循环队列的性质得出

--count; //出队成功,队列中元素数量减一

}

int top(){

if(count > 0){ //队列不为空的情况下才能获取队首元素

return data[head];

}

}

void output(){

for(int i = head;i != tail + 1;i = (i + 1)%length){ //从队首一直遍历到队尾,当遍历到最后一位时,跳转至数组起始位置

//特别注意此时循环截止的条件应该是i != tail + 1，因此可能tail的值小于head

cout << data[i] << " ";

}

cout << endl;

}

};

int main() {

Queue queue(100); //声明一个队列对象,并初始化

for(int i = 1;i <= 10;++i){

queue.push(i); //将1-10这10个数据元素依次插入队列中

}

queue.output(); //调用输出的方法

cout << "当前的队首元素为:" << queue.top() << endl;

queue.pop(); //出队

queue.output(); //调用输出的方法

cout << "出队操作后的队首元素为:" << queue.top() << endl;

return 0;

}

思考题：尝试用代码实现以下问题

1.能否使用两个堆栈实现一个队列？  
要求：只能使用栈的pop(),top()和push()，以及测试栈是否为空empty()四个操作. 来实现队列的empty(), push(),pop(),back(),front()等操作。

2.能否使用两个队列去实现一个堆栈？

请实现他的两个函数Push 和 Pop，分别完成入栈和出栈功能。

1.参考答案：

实现队列的pop方法弹出队头元素：stack是一个口进出，也不支持随机存取，所以你无法直接从栈底拿到第一个元素。要想拿到第一个元素，需要将左边栈容器中的所有元素拷贝到右边栈容器中。由于stack先进后出的数据结构，这样左边容器的第一个元素会作为最后一个元素存储到右边栈容器中，这样不就拿到左边栈容器中的第一个元素了吗 ?

实现队列的back方法获取队列的队尾元素：逻辑和思路1一样，唯一不同的是pop和top

//

// 利用两个栈实现⼀个队列.cpp

// c++

#include <iostream>

#include <stack>

using namespace std;

template <class T>

class MyQueue

{

public:

bool empty()const

{

return head.empty()&&tail.empty();

}

void push(T t)

{

head.push(t);

}

//删除对头元素

//因为queue是一种先进先出，而stack是先进后出，所以需要把head里的数据拷贝到tail中然后再从tail中pop头元素

void pop()

{

if(this->empty())

{

//throw exception("队列为NULL");

}

while(!head.empty())

{

tail.push(head.top());

head.pop();

}

//删除头元素

tail.pop();

//再将队尾栈容器元素拷贝到队头栈容器中

while(!tail.empty())

{

head.push(tail.top());

tail.pop();

}

}

T& back()

{

if(this->empty())

{

// throw exception("head is NULL");

}

return head.top();

}

//返回第一个元素

T& front()

{

if(this->empty())

{

//throw exception("队列为NULL");

}

while(!head.empty())

{

tail.push(head.top());

head.pop();

}

int tmp = tail.top();

//再将队尾栈容器元素拷贝到队头栈容器中

while(!tail.empty())

{

head.push(tail.top());

tail.pop();

}

return tmp;

}

private:

stack<int> head;

stack<int> tail;

};

int main()

{

MyQueue<int> q;

for(int i=1;i<5;i++)

{

q.push(i);

}

cout<<"front:"<<q.front()<<endl;

cout<<"back:"<<q.back()<<endl;

return 0;

}

2.参考答案：

题目分析：我们知道队列结构的特点是“先进先出”，栈结构的特点的“先进后出”；仔细分析这道题，本道题的要求为将两个队列组合起来形成一个“先进后出”结构。

这里我们令第一个队列为queue1，第二个队列为queue2，起始两个队列均为空。

首先我们陆续向栈中压入元素a、b、c，这里我们先压入queue1中，此时queue1中的元素为{a，b，c}（其中a为队首元素，c为队尾元素），queue2为空（如图a所示）；

接下来我们考虑从栈中弹出一个元素，依据栈“先进后出”的原则，我们第一次弹出的元素应该是c，但是c是queue1中的队尾元素，我们并不能直接从queue1获取到c元素,我们发现queue2此时为空，于是将queue1中的元素陆续弹出到queue2直至队尾元素留着队列中(如图b所示)；

接下来继续从栈中弹出一个元素,按照入栈顺序，应该出栈的是b，但是b是queue2的队尾元素，不能直接出栈,发现这个时候queue1是空的，于是将queue2中的元素依次出队到queue1直至队尾元素留在队列中(如图c所示)；

考虑继续插入一个元素d(此时考虑是插入queue1还是queue2)如果queue1为空，则插入queue2,如果queue2为空，则插入queue1，如果两个队列均为空，则回到初始状态,插入到queue1。此时queue1不为空，queue2为空，所以d插入queue1（如图queue1的元素为{a，d}，a为队首元素，d为队尾元素，queue2为空）；

接着弹出一个元素，依照入栈顺序，此时应该弹出元素d，依据上述方法，可得图e所示结果；

当queue1和queue2为空时，栈为空。

题目总结：

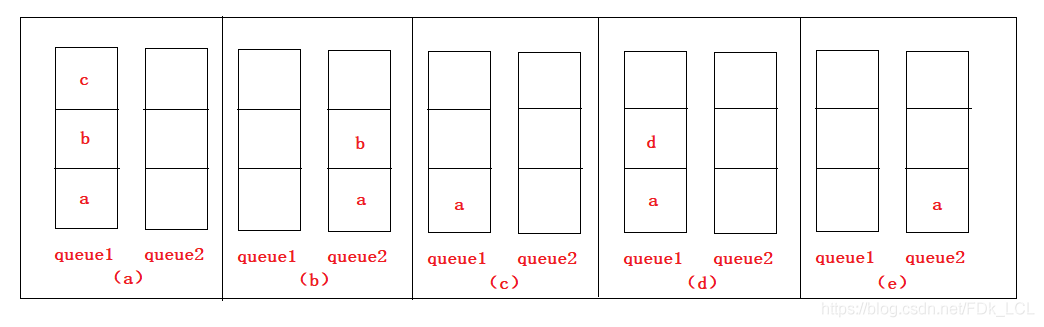
解决该题的关键主要在于两个关键：

<1>入栈元素应在放在那个队列中？？？

当两个队列均为空的时候，任意插入一个队列，当再次插入元素时，选择非空的队列进行插入；

<2>出栈时应该注意什么？？？

出栈时，将非空队列中的元素陆续出队并插入到空队列中直至非空队列只有队尾元素时；



代码如下：

#include<iostream>

#include<queue>

using namespace std;

template<typename T>

class CStack

{

public:

CStack(){}

~CStack(){}

void push(const T &val)

{

if(queue1.empty() && queue2.empty()) //两个队列为空，则插入queue1

{

queue1.push(val);

}

if(queue1.empty()) //如果queue1为空，则插入queue2

{

queue2.push(val);

}

else

{

queue1.push(val); //如果queue2为空，则插入queue1

}

}

T pop()

{

if(queue1.empty()) //如果queue1为空

{

if(queue2.empty()) //如果queue2为空，则说明栈为空

{

throw new exception("stack is empty");

}

else

{

if(queue2.size() == 1) //如过queue2只有一个元素，直接退出

{

T result = queue2.front();

queue2.pop();

return result;

}

else

{

while(queue2.size() != 1) //依次退出队中元素，直到队列中只有队尾元素

{

queue1.push(queue2.front());

queue2.pop();

}

T result = queue2.front(); //将队尾元素退出

queue2.pop();

return result;

}

}

}

else //如果queue1不为空

{

if(queue1.size() == 1) //如果queue1中只有队尾元素时，直接退出

{

T result = queue1.front();

queue1.pop();

return result;

}

else

{

while(queue1.size() != 1) //依次退出queue1中的元素到queue2，直到queue1只有队尾元素

{

queue2.push(queue1.front());

queue1.pop();

}

T result = queue1.front(); //将队尾元素弹出

queue1.pop();

return result;

}

}

}

private:

queue<T> queue1;

queue<T> queue2;

};

int main()

{

CStack<char> stack;

//测试用例1:

stack.push('a'); //元素a入栈

stack.push('b'); //元素b入栈

stack.push('c'); //元素c入栈

cout << "第1次出栈元素是：" << stack.pop() << endl;

cout << "第2次出栈元素是：" << stack.pop() << endl;

stack.push('d'); //元素d入栈

cout << "第3次出栈元素是：" << stack.pop() << endl;

cout << "第4次出栈元素是："<< stack.pop() << endl;

return 0;

}