队列讲稿

一、队列的逻辑结构

引例、队列及其操作（P1500）

队列（queue）：在线性表的一端插入元素，在另一端删除元素，所以遵循先进先出原则，其中删除元素的一端称为队首(front)，插入元素的一端称为队尾（rear）。队列就像我们排队打饭，先来的先打饭，后来的只能排在队尾。

【输入格式】

第一行包含一个整数 N（N<=20000），表示有 N 条关于 queue 的操作，在进行任何操作之前，queue 都是空的。接来的 N 行，每行是一个关于 queue 的操作，格式和含义如下：

clear： 把队列置空。　　**empty：** 判断队列是否为空。

push x： 把整数 x 插入队尾（x 为 int 范围里的数）。　　**pop：** 队首元素出队列。

front： 获取队首元素的值。

【输出格式】

对于 front 操作，输出一个整数，如果这个操作失败，则输出单词 error。

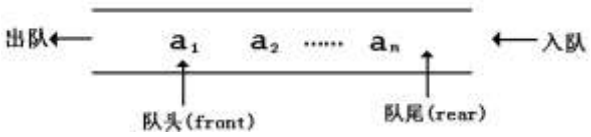
对于 pop 操作，如果这个操作失败，则输出单词 error。

对于 empty 操作，如果队列是空，则输出"empty"，否则输出"not empty'。

【讲解】

1、队列的逻辑结构

题目中对栈和队列的逻辑结构和操作规则定义的非常清楚。图形描述如下：



队列满足"先进先出（FIFO）"的原则。它的运行机制是：元素从队尾进，队首出，不允许插队！

2、STL 队列容器和数组模拟队列

STL 队列容器	数组队列
#include<queue> 定义格式： queue<元素类型>队列名称 例如： queue<int>q;	const int SIZE=20005　//队列的容量 元素类型 q[SIZE];　//数组用于存储队列元素 int front,rear;　//队首指针，队尾指针 说明： ◆ 队列元素的类型随着不同问题而不同 ◆ 变量 front：是队首指针，始终指向队列第一个元素位置 ◆ 变量 rear：　队尾指针，始终指向队尾元素后一个位置 ◆ 当 front==rear 时，队列为空，否则队列为非空 ◆ 队列中如果有 n 个元素，则需要 n+1 的空间，因为 rear 始终指向空元素。

3、栈和队列的运算

运算	STL 容器元算函数	数组队列运算代码
计算队列中元素个数	int num=q.size();	int num=rear-front;
empty：判断队列是否为空	if(q.empty()) 则队列为空	如果 front==rear，则队列为空
push x ：把整数 x 插入队尾	q.push(x);	queue[rear++]=x;
pop：　队首元素出队列。	q.pop();	front++;
clear：把队列置空	while(!q.empty()) q.pop();	front=rear=0;
front：读取首元素的值。	x=q.front();	x=queue[front]

4、本题算法流程

数组队列操作

```
int q[20005];
int front,rear;
void solve2()
{
   char op[10];
   scanf("%d",&N);
   front=rear=0;    //初始化队列
   for(int i=1;i<=N;i++)
   {
       scanf("%s",op);
       if(strcmp(op,"clear")==0)  //清空
       {
          front=rear=0;
       }
       else if(strcmp(op,"empty")==0) //判空
       {
          if(front==rear)
              printf("empty\n");
          else
              printf("not empty\n");
       }
       else if(strcmp(op,"push")==0) //入队
       {
          scanf("%d",&x);
          queue[rear++]=x;
       }
       else if(strcmp(op,"pop")==0) //删除队首
       {
           if(front==rear)
              printf("error\n");
           else
              front++;
       }
       else if(strcmp(op,"front")==0) //读队首
       {
           if(front==rear)
             printf("error\n");
           else
           {
              x=queue[front];
              printf("%d\n",x);
           }
       }
    }
}
```

STL 队列容器操作

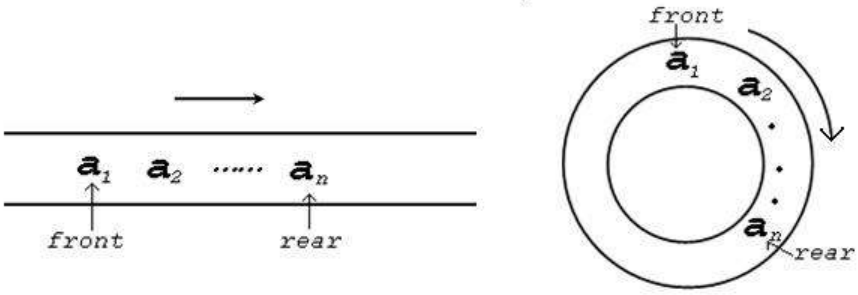
```
void solve2()
{
    char op[10];
    scanf("%d",&N);
    queue<int>q;     //初始化定义队列
    for(int i=1;i<=N;i++)
    {
        scanf("%s",op);
        if(strcmp(op,"clear")==0)
        {
           while(!q.empty())q.pop();
        }
        else if(strcmp(op,"empty")==0)
        {
           if(q.empty())
               printf("empty\n");
           else
               printf("not empty\n");
        }
        else if(strcmp(op,"push")==0)
        {
           scanf("%d",&x);
           q.push(x);
        }
        else if(strcmp(op,"pop")==0)
        {
           if(q.empty())
               printf("error\n");
           else
               q.pop();
        }
        else if(strcmp(op,"front")==0)
        {
           if(q.empty())
               printf("error\n");
           else
           {
              x=q.front();
              printf("%d\n",x);
           }
        }
    }
}
```

二、队列应用例题

例 1、循环队列（P1504）

用数组模拟队列操作，可以类比成：队首指针 front 始终在一条直线追赶队尾指针 rear，当 front 与 rear 相遇时（front==rear），队列为空，称这样的队列为**线性队列**。在这个过程中，front 走过的地方，再也不会走到，造成极大的空间浪费。

如果把模拟队列的数组看成一个首尾相接的圆环，此时的队列操作，就是 front 在一个圆环上追赶 rear，称这样的队列为**循环队列**。显然 front 走过的地方，可以多次重复走到。



线性队列　　循环队列

由于 front 和 rear 在圆环上追赶，当 front 与 rear 相遇（front==rear）时，队列是空还是满呢？不好判断，因为有可能 rear 比 front 多跑一圈。所以，我们在定义循环队列时，需要特别注意数组的大小，一般的要先分析队列中可能的最多元素个数，数组的大小应比最多元素个数大，这时候才能用 front==rear 判断队列为空。

现在的问题是：我们给出一个循环队列的空间限制 n，然后给出 m 个进队和出队的操作，请你输出最后的队列中的元素（按元素依次出队的顺序）。要特别说明的是，根据数组队列的操作规则，rear 始终是指向队尾元素的后面的位置，所以给出循环队列空间为 n，则实际只能使用 n-1 个空间。另外，由于空间的限制，当 rear 与 front 相遇后（溢出），应丢弃队首元素，为 rear 腾空间！

【样例】

4 //队列占用空间 n 7 //操作数 k in 1 in 2 in 5 in 6 out out in 8	6 8 //最后队列中元素

【讲解】

1、数组环形队列的操作代码：

入队操作	q[rear]=x;
	rear=(rear+1)%n; //循环
	if(rear==front) //溢出后
	front=(front+1)%n;
		队列中元素个数	(rear-front+n)%n

出队操作	if(front!=rear) front=(front+1)%n;	**判断队列是否为空**	if(front==rear) 为空;
读取队首元素	x=q[front]		

2、本题代码框架

```
int main()
{
      输入数 n,k 并初始化队列为空;
      for(int i=1;i<=k;i++)
      {
         scanf("%s",op);
         if(op[0]=='i')  入队操作;       //in 操作
         else if(op[0]=='o') 出队操作  //out 操作
      }
      若队列为空，则输出-1；否则输出队列中的元素;
    return 0;
}
```

一点说明：STL 队列不存在数组队列的缺点，因为它每次 pop 之后，是真实地删除队首元素，也就是说，是物理上的永久删除，不会造成空间上的浪费。

例 2、卡片游戏（P1501）

桌子上有一叠牌，从第一张牌（即位于顶面的牌）开始从上到下依次编号为 1..N，当至少还剩下两张牌时进行以下操作：把第一张扔掉，然后把新的一张放到整叠牌的最后。

输入 N，输出每次扔掉的牌，以及剩下的牌。

【样例】

输入	输出
7	1 3 5 7 4 2 6

【数据范围】 N<=20000

【讲解】

把这叠牌看成一个队列，然后按如下进行模拟：

1、初始队列中元素从头到尾依次为：1,2,…,N;

2、扔掉一张：看成是删除队首元素;

3、把新的一张放到整叠牌的最后：看成把队首元素插入到队尾。

重复 2、3 操作直到队列为空。

```
int main()
{
      输入 N;
      建立初始队列：1,2,…,N;
      while(队列不为空)
      {
          读取队首元素，并输出 ;
          删除队首元素;              //扔卡片
          读取取队首元素并把该元素插入队尾 ;
          删除队首元素;              //放到牌叠最后
      }
    return 0;
}
```

时间复杂度为：O(2*n)

思考：如果用数组线性队列的容量 SIZE 应设多的？ 使用数组循环队列是否更好？

当然，STL 队列容器不存在容量的问题！

例 3、约瑟夫问题[1]（P1527）

已知 n 个人（编号分别为 1,2,3,…,n）围坐在一张圆桌周围。从编号为 1 的人开始报数，数到 m 的那个人出列；他的下一个人又从 1 开始报数，数到 m 的那个人又列出……，依此规律重复下去，直到圆桌周围的人全部出列。

【输入格式】

两个整数 n，m

【输出格式】

按顺序给出的出列人的编号。

【样例】

9 5	5 1 7 4 3 6 9 2 8

【数据范围】

2<= m < n <= 1000

【讲解】

类似上题，同样可以用队列模拟报数的过程：

```
初始化队列;
1..n 依次进队;
while(队列不为空)
{
        for(int i=1;i<m;i++)  //从队首开始报 1..m-1
        {
            读取队首元素到 x 中;
            删除队首元素（pop）;
            把 x 元素插入队尾（push）;
        }
        输出队首元素并输出;   //该元素报数 m
        删除队首元素;
}
```

时间复杂度：O(n*m)

请思考：

1、用线性队列，队列空间要设置多大

2、循环队列，空间设置多大

3、STL 队列的好处！

三、关于队列的信息学初赛试题

1. 队列 Q 的对首指针 front=3，队尾指针 rear=10，则队列中元素的个数为（ ）

A、6 B、7 C、8 A、10

2. 循环队列 Q 的下标范围为 1..N，队首指针为 front，队尾指针为 rear，则队列中元素的个数为（ ）

A、rear-front B、rear-front+1 C、(rear-front+1)%N A、(rear-front+N)%N

3. 设栈 S 和队列 Q 的初始状态为空，元素 e_1，e_2，e_3，e_4，e_5，e_6 依次通过栈 S，一个元素出栈后即进入队列 Q，若出队的顺序为 e_2，e_4，e_3，e_6，e_5，e_1，则栈 S 的容量至少应该为（ ）。

A、2 B、3 C、4 D、5

4. (08 年)已知队列（13，2，11，34，41，77，5，7，18，26，15），第一个进入队列的元素是 13，则第五个出队列的元素是（ ）。

A. 5 B. 41 C. 77 D. 13 E. 18