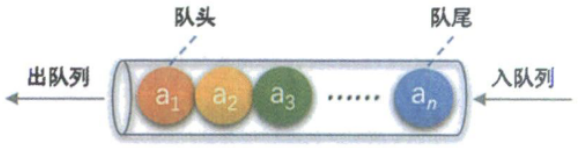
05. 队列

5.1 队列的定义

队列（queue）是只允许**在一端进行插入**操作，而在**另一端进行删除**操作的线性表

队列是一种先进先出（First In First Out）的线性表，简称FIFO。允许插入的一端称为**队尾**，允许删除的一端称为**队头**。



5.2 存储方式

```
template <typename T>
class LoopQueue
{
public:
    LoopQueue(int c = 10);
    ~LoopQueue();

    bool isEmpty();      //队列的判空
    int getSize();       //获取队列的大小
    void push(T t);      //入堆
    void pop();          //出队
    T getFront();        //获取队头

private:
    int capacity;
    int begin;
    int end;
    T* queue;
};
```

变量解释：

- `begin`：表示队列头部的索引。
- `end`：表示**下一个元素插入的位置**的索引。
- `capacity`：数组容量
- `queue`：数组

5.3 队列的顺序存储（循环链表）

5.3.1 循环队列的定义

队列的头尾相接的顺序存储结构称为循环队列

初始时`end = begin = 0`

队列空条件： `end == begin`

队列满条件： `(end + 1) % capacity == begin`

5.3.2 循环队列的入队

思路：

1. 先判断队列是否已满`(end + 1) % capacity == begin`则队列满
2. 未满则将数据存入`queue[end]`
3. `end`移向下一位也就是`end + 1`，但是这是循环链表，如果到了数组的结尾则要到开头去，对`end + 1`取模（`capacity`）可以保证end在数组范围之内，即`end = (end + 1) % capacity;`

```
template<typename T>
void LoopQueue<T>::push(T t)
{
    if ((end + 1) % capacity == begin)                //判断队列是否已满   end + 1 %
```

```
capacity 是插入完这个元素之后，下一个要插的位置，因为是循环队列，所以要 % capacity来保证在数组范围之内
    {
        puts("队列已满！");
    }
    else
    {
        queue[end] = t;
        end = (end + 1) % capacity;          //如果是无限长的队列，那么end = end + 1，但是这是循环队列，要保证end在数组范围之内，所以要 % capacity
    }
}
```

5.3.3 循环队列的出队

思路：

1. 判断队列是否为空（`end == begin`）
2. 如果不为空则队头`begin`往下一位移一位，也就是`begin + 1`，但是这是循环链表，如果到了数组的结尾则要到开头去，对`begin + 1`取模（`capacity`）可以保证end在数组范围之内，即`begin = (begin + 1) % capacity;`

```
template<typename T>
void LoopQueue<T>::pop()
{
    if (end == begin)       //判断队列是否为空
    {
        puts("队列为空！");
    }
    else
    {
        begin = (begin + 1) % capacity;     //队头往下一个位置进一位
    }
}
```