数据结构

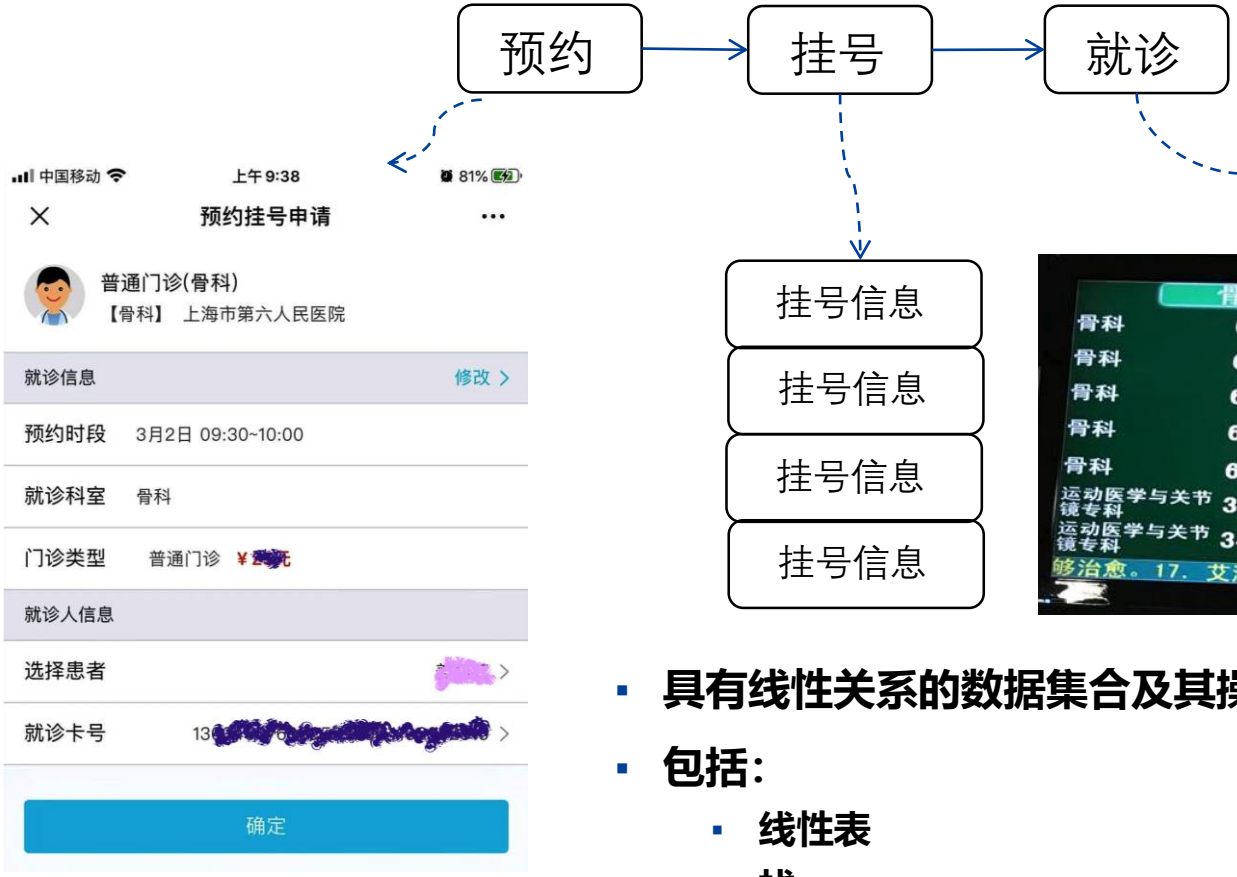
教师：姜丽红 IST 实验室 jianglh@sjtu.edu.cn

助教：芮召普、 江嘉晋，IST实验室 软件大楼5号楼5308

《软件基础实践》教师：杜东 IPADS 实验室 软件大楼3号楼4层

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

队列



- **具有线性关系的数据集合及其操作**
- **包括:**
  - **线性表**
  - **栈**
  - **队列**
  - **字符串（不讲）**

排队系统模拟

- 模拟 / 仿真（simulation）是计算机的一个重要应用，是指用计算机来仿真现实系统的操作并收集统计数据。
- 例如，有K个出纳员的银行系统，以确定要提供合理的服务时间，最小的K值是多少。
- 计算机仿真有很多优点：
  - 首先，不需要真实的顾客就能够得到统计信息；
  - 第二，由于计算机速度很快，使用计算机模拟比真实的实现要快很多；
  - 第三，模拟结果可以简单地重现。

离散事件模拟系统

- 一个排队系统主要由一些离散事件组合而成。
- 在银行的排队系统中主要有两类事件：顾客的到达和服务完毕后顾客的离去。
- 整个系统就是不断地有到达事件和离开事件发生，这些事件并不是连续发生的，因此这样的系统被称为离散事件模拟系统
- 一个离散事件模拟器由事件处理组成；一般有两类事件：
  - 客户到达
  - 服务完毕，客户离开
- 可以在模拟过程中统计客户的排队长度、等待时间、服务员的连续工作时间、空闲时间等统计信息。

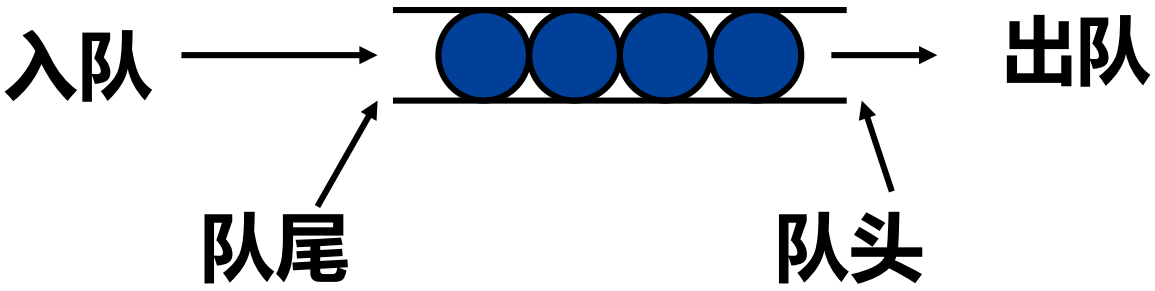
第四章 队列

- **队列的概念**
- **队列的实现**
- **队列的应用**

本章学习目标
1.分析应用对象，利用队列进行问题建模。
2.实现创建、进队、出队以及销毁操作。

队列

- **队列是一种常用的线性结构，到达越早的结点，离开的时间越早。所以队列通常称之为先进先出(FIFO: First In First Out)队列。**



打印队列管理等

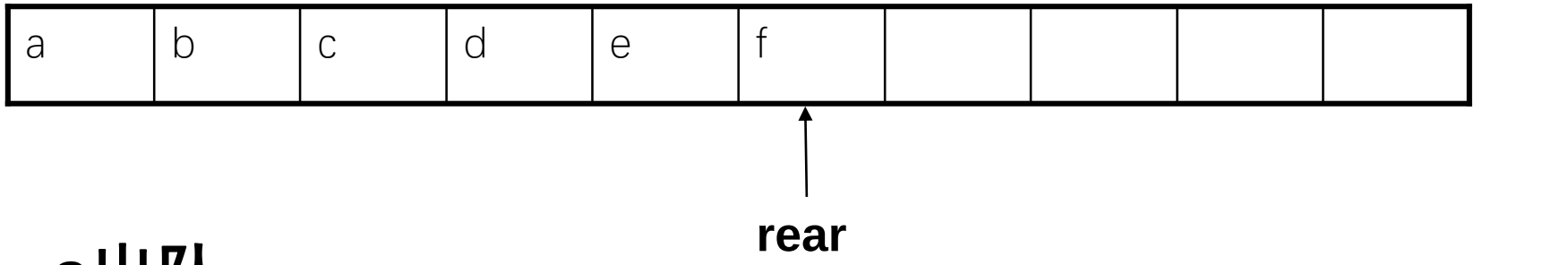
队列的抽象类

```
template <class elemType>
class queue
 { public:
    virtual bool isEmpty() = 0;                    //判队空
    virtual void enQueue(const elemType &x) = 0; //进队
    virtual elemType  deQueue() = 0;               //出队
    virtual elemType getHead() = 0;               //读队头元素
    virtual ~queue() {}                          //虚析构函数
 };
```

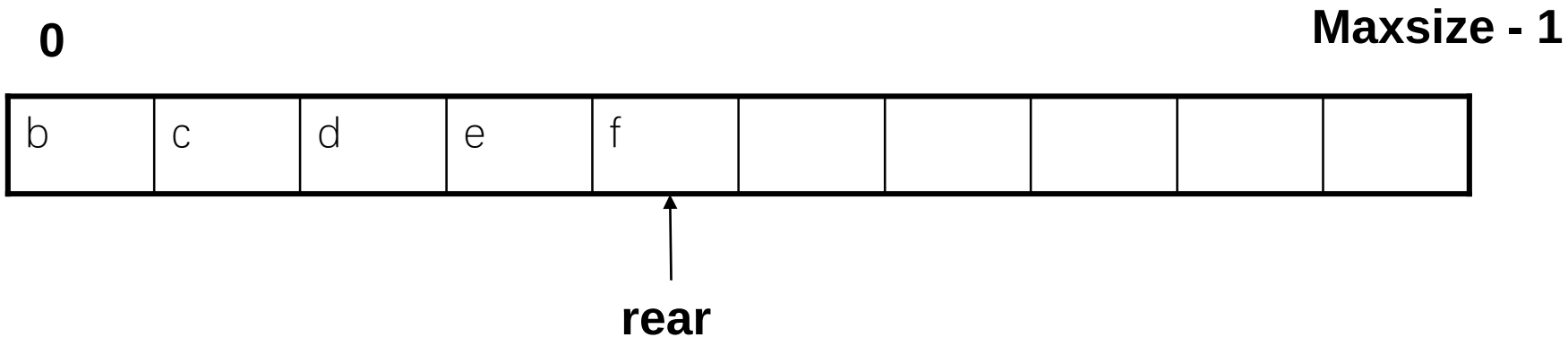
队列的顺序存储实现

- 使用数组存储队列中的元素；
- 队列中的结点个数最多为 MaxSize个；
- 元素下标的范围从0到MaxSize-1。
- 顺序队列的三种组织方式
  - 队头位置固定
  - 队头位置不固定
  - 循环队列

队头位置固定

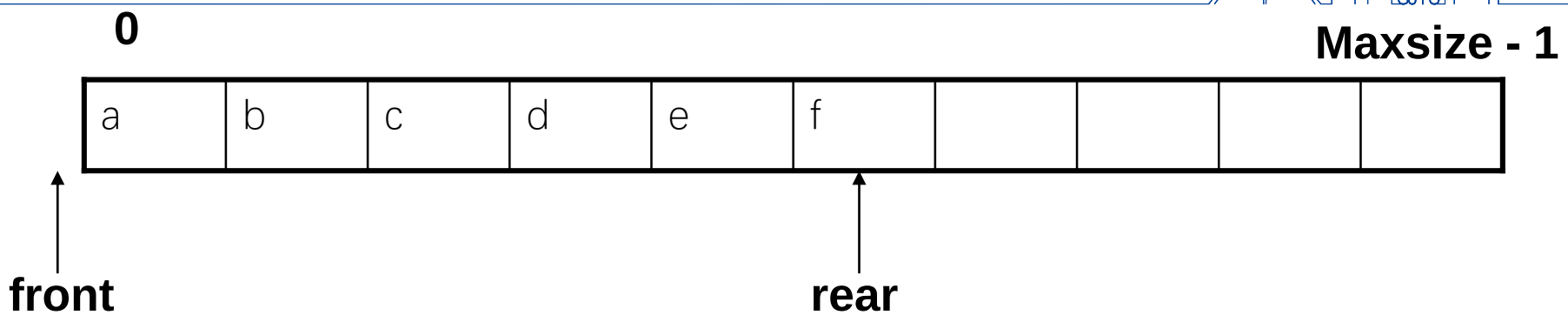


a出队

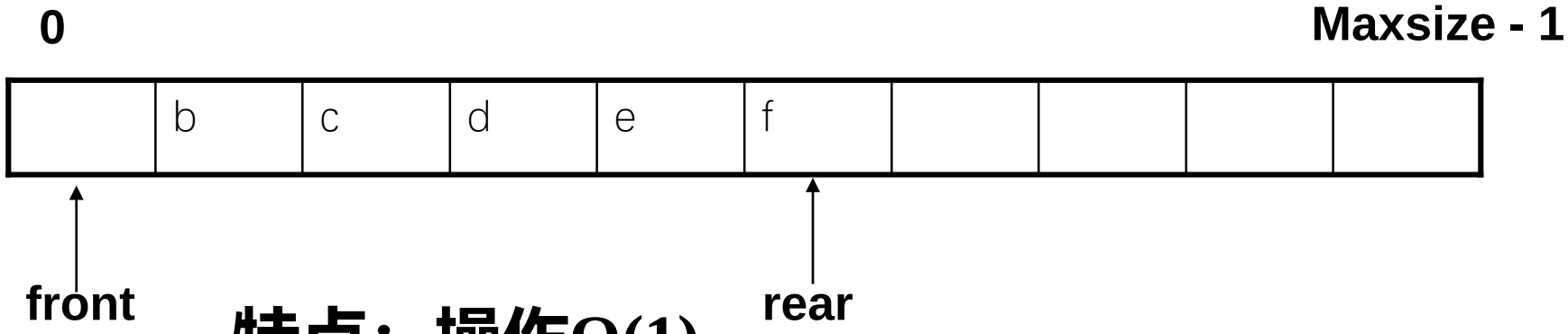


缺点：出队会引起大量的数据移动

队头位置不固定

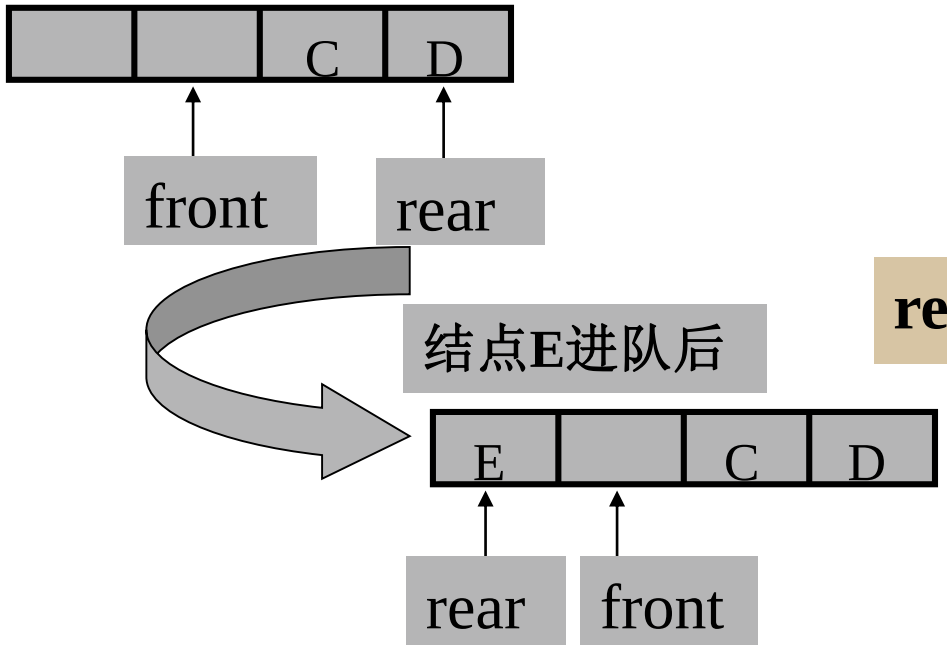


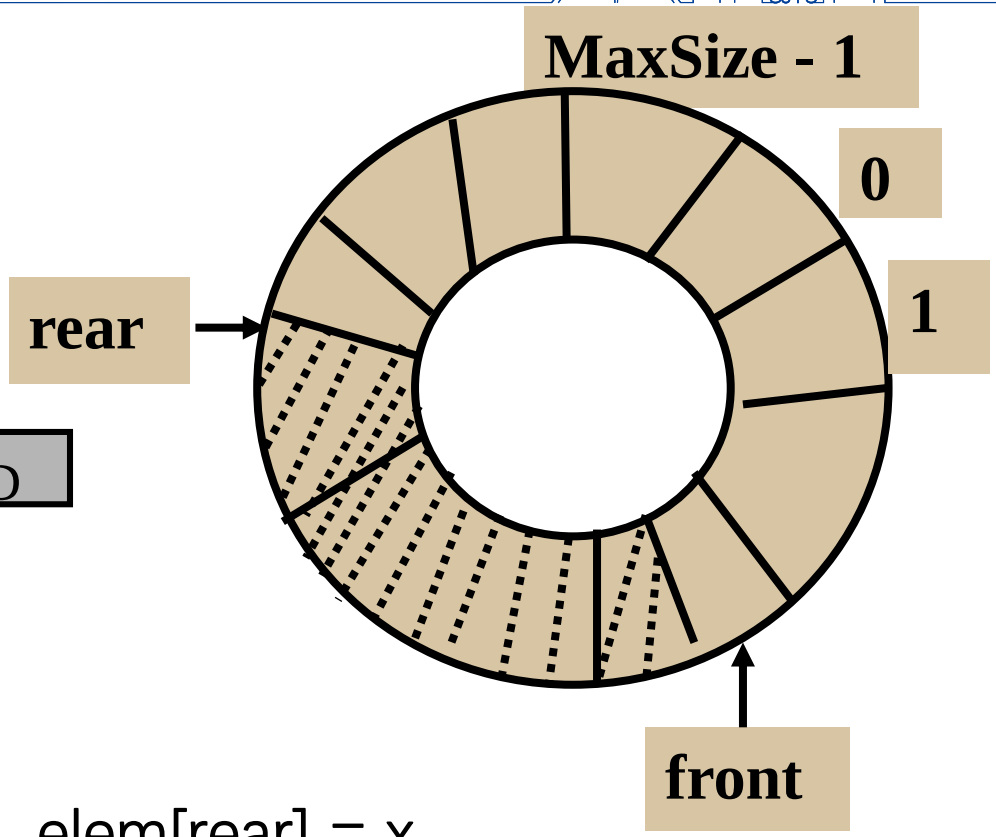
a出队



特点：操作O(1)
浪费空间

循环队列





入队操作：

rear = (rear + 1) % MaxSize ； elem[rear] = x

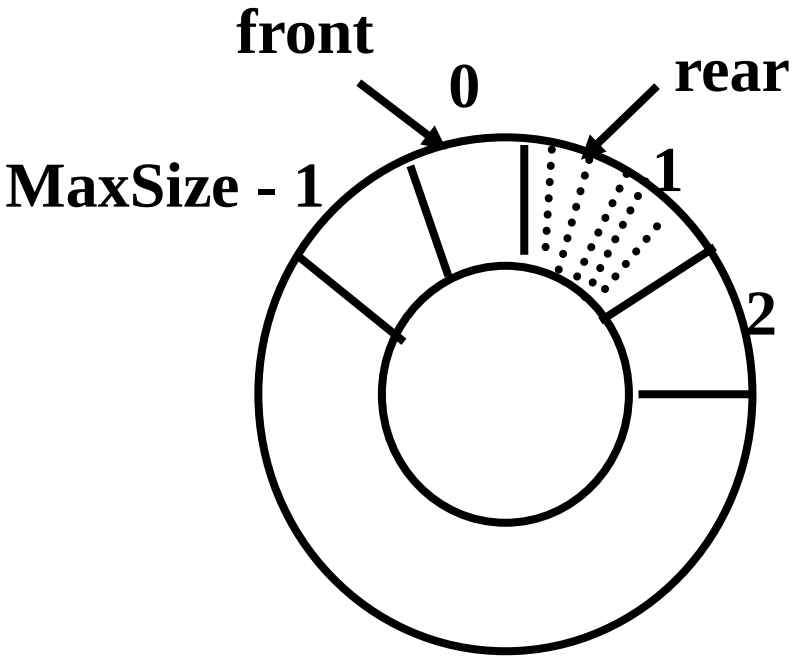
出队操作：

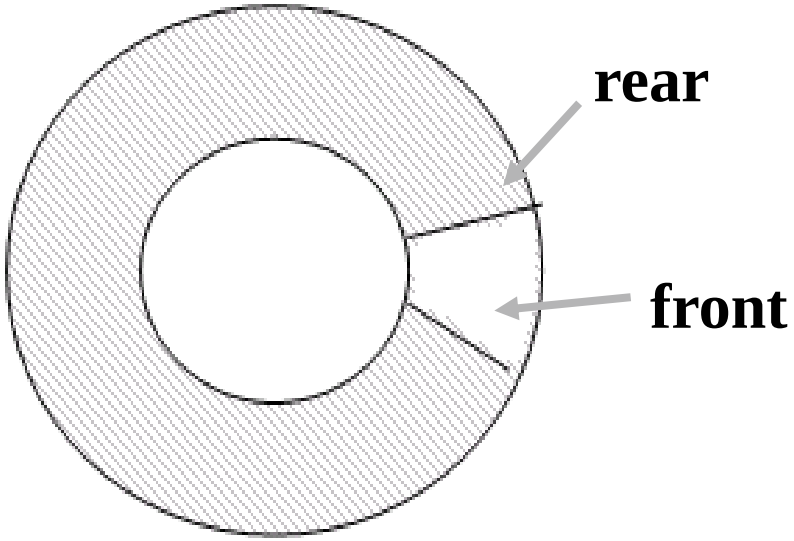
front = (front + 1) % MaxSize

循环队列空与队列满的问题

- 最后一个元素出队时，执行
- front = (front + 1) % MaxSize，front和rear相同。
- 因此队列为空时，front== rear。

只剩最后一个空位置，执行入队操作
rear = (rear + 1) % MaxSize
与front重叠，front==rear。

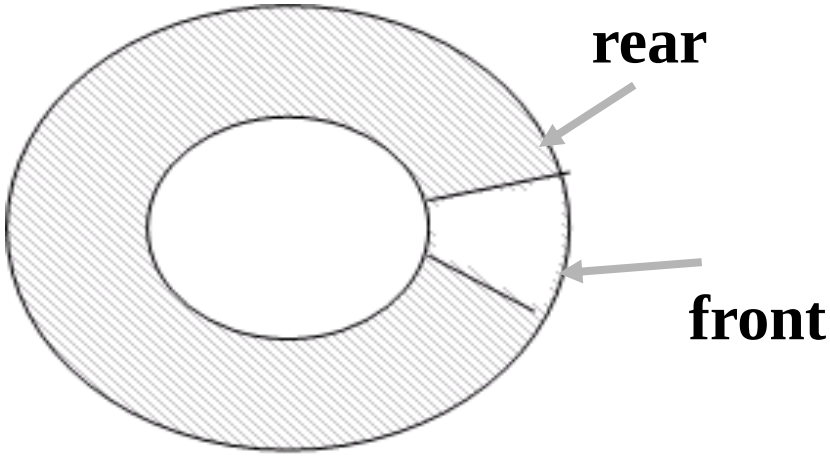




循环队列front除了初始状态时，其余时间不一定指向0号下标。Front和rear“你逃我追”

解决方案

- “牺牲”一个单元，规定front指向的单元不能存储队列元素，只起到标志作用，表示后面一个是队头元素。
- 当rear“绕一卷”赶上front时，队列就满了。因此队列满的条件是：(rear + 1) % MaxSize == front
- 队列为空的条件是front == rear，即队头追上了队尾。



循环队列类的定义


```
template <class elemType>
class seqQueue: public queue<elemType>
{private:
      elemType *elem;
        int maxSize;
        int front, rear;

        void doubleSpace();
```

```
public:
    seqQueue(int size = 10);
    ~seqQueue();
    bool isEmpty();
    void enQueue(const elemType &x);
    elemType deQueue();
    elemType getHead();
};
```

构造函数

- **申请一块空间，首地址存入elem。数组规模保存在MaxSize中，front和rear置成0。**

```
template <class elemType>
  seqQueue<elemType>::seqQueue(int size)
{
        elem = new elemType[size];
        maxSize = size;
        front = rear = 0;
}
```

析构函数

- **释放存储队列的空间**

```
template <class elemType>
seqQueue<elemType>::~seqQueue()
{
    delete [] elem;
}
```

enQueue函数

- 首先要判断数组是否已放满，需要时扩大数组空间。
- 将元素放入第一个空位，队尾向后移。

```
template <class elemType>
void seqQueue<elemType>::enQueue
                          (const elemType &x)
{
    if ((rear + 1) % maxSize == front)
                doubleSpace();
    rear = (rear + 1) % maxSize;
    elem[rear] = x;
}
```

doubleSpace

注意和线性表的doubleSpace的不同之处

```
template <class elemType>
void seqQueue<elemType>::doubleSpace()
{    elemType *tmp =elem;
     elem = new elemType[2 * maxSize];
     for (int i = 1; i < maxSize; ++i)
          elem[i] = tmp[(front + i) % maxSize];

     front = 0; rear = maxSize - 1;
     maxSize *= 2;
     delete tmp;
}
```

deQueue函数

- 将front往后移一个位置
- 返回elem [front]的内容。

```
template <class elemType>
    elemType seqQueue<elemType>::deQueue()
{
        front = (front + 1) % maxSize;
        return elem[front];
}
```

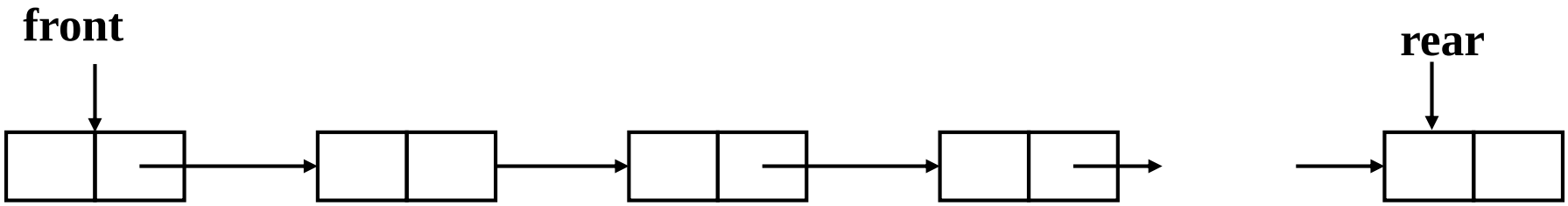
isEmpty函数

- **判断**front**是否等于**rear**。若相等，返回**true**，否则返回**false**。**

```
template <class elemType>
bool seqQueue<elemType>::isEmpty() const
{
      return front == rear;
}
```

队列的链式实现

用无头结点的单链表表示队列，表头为队头，表尾为队尾



链式队列类的定义


```
template <class elemType>
class linkQueue: public queue<elemType>
 { private:
       struct node {
        elemType  data;
        node *next;
        node(const elemType &x, node *N = NULL)
                  {   data = x; next = N;}
        node():next(NULL) {}
        ~node() {}
      };
      node *front, *rear;
```

链式队列类的定义


```
public:
    linkQueue();
    ~linkQueue();
    bool isEmpty() ;
    void enQueue(const elemType &x) ;
    elemType deQueue() ;
    elemType getHead() ;
};
```

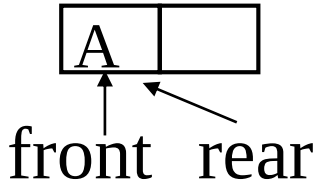
链接队列的特点

- 链接队列不会出现队列满的情况，但队列为空的情况依然存在。
- 队列为空时，单链表中没有结点存在，即头尾指针都为空指针。
- 保存一个链接队列只需要两个指向单链表结点的指针front和rear，分别指向头尾结点。
- 采用链接存储时，队列的基本运算的实现也非常简单，都是O（1）的时间复杂度。

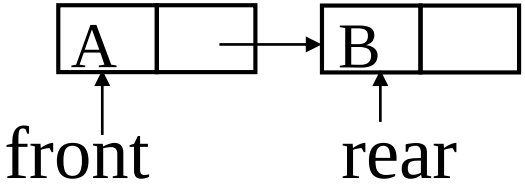
队列操作的实例

初始时：front= NULL; rear = NULL;

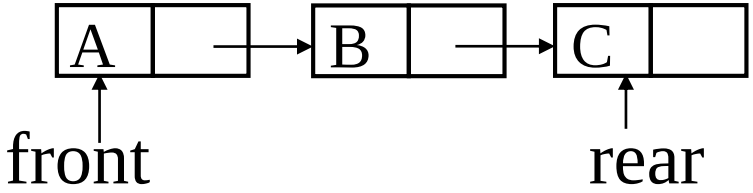
A进队：



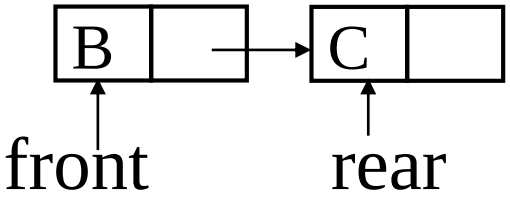
B进队：



C进队：



出队：



构造函数

- **将front和rear设为空指针。**

```
template <class elemType>
linkQueue<elemType>::linkQueue()
{
    front = rear = NULL;
}
```

enQueue(x)

```
template <class elemType>
void linkQueue<elemType>::enQueue(const
elemType &x)
{
    if (rear == NULL)
        front = rear = new node(x);
    else
        rear = rear->next = new node(x);
}
```

deQueue()

```
template <class elemType>
elemType linkQueue<elemType>::deQueue()
{//空队列处理?
    node *tmp = front;
    elemType value = front->data;
    front = front->next;
    if (front == NULL) rear = NULL;// 最后一个元素出队
    delete tmp;
    return value;
}
```

getHead()和isEmpty()


```
template <class elemType>
bool linkQueue<elemType>::isEmpty() const
{
    return front == NULL;
}

template <class elemType>
elemType linkQueue<elemType>::getHead() const
{
    return front->data;
}
```

析构函数


```
template <class elemType>
linkQueue<elemType>::~linkQueue()
{
    node *tmp;
    while (front != NULL) {
        tmp = front;
        front = front->next;
        delete tmp;
    }
}
```

顺序实现和链接实现的比较

- **时间：两者都能在常量的时间内完成基本操作，但顺序队列由于采用回绕，使入队和出队的处理比较麻烦**
- **空间：链接队列中，每个结点多一个指针字段，但在顺序队列中有大量的尚未使用的空间。**

排队系统的实现

- 实现一个排队系统就是模拟这两类事件的生成和处理。
- 事件处理过程
  - 当遇到一个到达事件时，表示有一个新顾客到达。如果服务员没空，顾客到队列去排队。否则为这个顾客生成服务所需的时间t。经过了t 时间以后会产生一个顾客离开事件。
  - 当遇到一个离开事件时，就检查有没有顾客在排队。如果有顾客在排队，则让队头顾客离队，为他提供服务。如果没顾客排队，则服务员可以休息

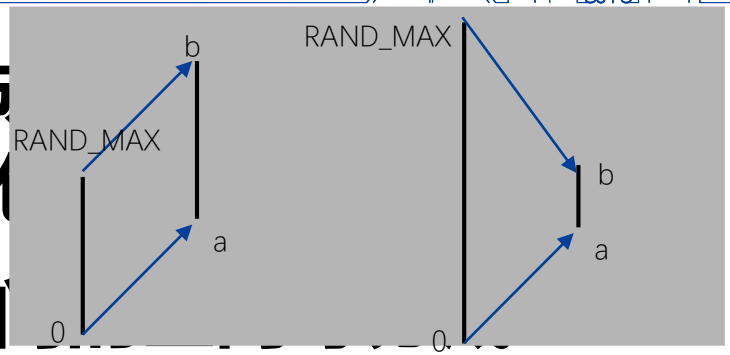
如何产生顾客到达事件和服务时间

- 顾客的到达时间间隔和为每个顾客的服务时间并不一定是固定的。
- 尽管服务时间和顾客到达的间隔时间是可变的，但从统计上来看是服从一定的概率分布。
- 要生成顾客的到达时间或生成服务时间必须掌握如何按照某个概率生成事件。

均匀分布的概率事件

- 用随机数产生器产生的随机数
- 如顾客到达的间隔时间服从[a，b]之间的均匀分布，则可以生成一个[a，b]之间的一个随机数x，表示前一个顾客到达后，经过了x的时间后又有一个顾客到达了。

[a，b]之间的随机数的产生



- 假如系统的随机数生成器生成
布在0到RAND_MAX之间，
- 把数轴上0到RAND_MAX之
份；当生成的随机数落在第一个区间中，则表示生成的是a；当落在第二个区间中时，表示生成的是a+1;……，当落在最后一个区间时，表示生成的是b。
- 这个转换可以用一个简单的算术表达式rand() * (b – a + 1)/(RAND_MAX + 1) + a实现。

虚拟时间

- 模拟系统是一个虚拟的系统。当我们得到了在x秒后有一个事件生成的信息时，并不真正需要让系统等待x秒，然后处理该事件。
- 在模拟系统中，一般不需要使用真实的精确时间，只要用一个时间单位即可，我们把这个时间单位叫做一个嘀嗒。
- 一个嘀嗒可以表示1秒，也可以表示1分钟，也可以表示一小时，这根据应用来决定。

离散的时间驱动模拟

- 沿着时间轴，模拟每一个嘀嗒中发生了什么事件并处理该事件。
- 模拟开始时时钟是0 嘀嗒，随后每一步都把时钟加1 嘀嗒，并检查这个时间是否有事件发生；如果有事件发生，我们就处理该事件并生成统计信息；当到达规定时间时，模拟结束。

缺陷与解决方法

- **离散的时间驱动模拟连续地处理每个时间单位；这种模拟对于时间间隔很大的事件很不适合。如果在很长的一段时间中没有任何事情发生，程序还必须检查每个嘀嗒中是否有事件发生。这将浪费计算机的时间。**
- **解决方法：**事件驱动的模拟
- **将事件按照发生时间排队，当一个事件处理结束后，直接将时间拨到下一事件的发生时间，处理下一事件。**

银行排队系统的模拟系统

- **设计一个最简单的排队系统模拟器，即只有一个服务台的排队系统，希望通过这个模拟器得到顾客的平均排队时间。**
- **银行中只有一个服务台，顾客到达的时间间隔服从[arrivaLow, arrivalHigh]的均匀分布，服务时间长度服从[serviceTimeLow, serviceTimeHigh]间的均匀分布，一共模拟customNum个顾客。要求统计顾客的平均排队时间。**

单服务台的排队系统的思路

- 使用一个辅助队列
- 整个模拟由三个步骤组成：
  - 首先生成所有的顾客到达事件，按到达时间排成一个队列；
  - 服务员一旦有空，就为队头元素服务，在提供服务前先检查该顾客等待了多少时间，记入累计值；
  - 最后，在所有顾客都服务完以后，返回累计值除以顾客数的结果。

totalWaitTime = 0;
设置顾客开始到达的时间currentTime = 0;
for (i=0; i<customNum; ++i)
{ 生成下一顾客到达的间隔时间;
下一顾客的到达时间currentTime
+= 下一顾客到达的间隔时间;
将下一顾客的到达时间入队;
}
从时刻0开始模拟;
while (顾客队列非空)
{ 取队头顾客;
If (到达时间 > 当前时间)
直接将时钟拨到事件发生的时间;
Else 收集该顾客的等待时间;
生成服务时间;
将时钟拨到服务完成的时刻; }
返回 等待时间/顾客数;

模拟类的定义

- 设计一个实现单服务台排队系统的工具。
- 数据成员：到达时间间隔的分布，服务时间的分布，以及想要模拟多少个顾客。
- 功能：获得本次服务中顾客的**平均等待时间**是多少。这个类应该有两个公有函数：构造函数接受用户输入的参数，另外一个就是执行模拟并最终给出平均等待时间的**函数avgWaitTime**。

```
class simulator{
   int arrivalLow;
   int arrivalHigh;
   int serviceTimeLow;
   int serviceTimeHigh;
   int customNum;
public:
   simulator();
   int avgWaitTime() const;
};
```

构造函数

```
simulator::simulator() {
    cout << "请输入到达时间间隔的上下界： ";
    cin >> arrivalLow >> arrivalHigh;
    cout << "请输入服务时间的上下界： ";
    cin >> serviceTimeLow >> serviceTimeHigh;
    cout << "请输入模拟的顾客数： ";
    cin >> customNum;
    srand(time(NULL));//利用系统时间初始化随机数发生器种子，即随机数发生器初始化
}
```

avgWaitTime 平均等待时间

```
int simulator::avgWaitTime() const
{ int currentTime = 0;   //当前时间
  int totalWaitTime = 0;  //总的等待时间
  int eventTime;
  linkQueue<int> customerQueue;
  int i;
  for (i=0; i<customNum; ++i)
       { currentTime += arrivalLow +
              (arrivalHigh -arrivalLow +1) * rand() /
               (RAND_MAX + 1);
         customerQueue.enQueue(currentTime);
        }
```

```
currentTime = 0;
  while (!customerQueue.isEmpty())
      { eventTime = customerQueue.deQueue();
        if (eventTime < currentTime)
            totalWaitTime += currentTime - eventTime;
        else currentTime = eventTime;
       currentTime += serviceTimeLow +
                  (serviceTimeHigh -serviceTimeLow +1)
                  * rand() / (RAND_MAX + 1);
      }
   return totalWaitTime / customNum;
}
```

simulator类的应用


```
#include "simulator.h"
#include <iostream>
using namespace std;
int main()
{
  simulator sim;
  cout << "平均等待时间：  “
       << sim.avgWaitTime() << endl;
  return 0;
}
```

某次执行结果

请输入到达时间间隔的上下界：0 5

请输入服务时间的上下界：1 4

请输入模拟的顾客数：1000

平均等待时间：18

优先级队列(第六章)

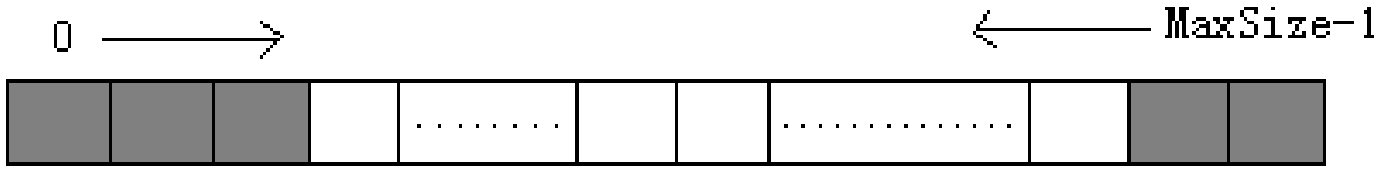
- 结点之间的逻辑关系是由优先级决定。优先级高的先出队，优先级低的后出队。
- 利用现有的队列结构。有两种方法实现优先级队列。
- （1） 入队时，按照优先级在队列中寻找正确的位置。
- 入队操作O(N)，出队操作O(1)
- （2）入队时，放队列尾。出队时按照优先级别。
- 入队操作O(1)，出队操作O(N)。
- Summary:
- 队列是一种先进先出的线性表
- 队列可以是顺序实现、链接实现
- 队列是应用很广的一种数据结构

作业（以下5道题计算平时成绩）3月27日

- 1、试写出求循环队列长度的表达式
- 2、循环队列的优点是什么？如何判断队空和队满？
- 3、若以一个大小为6的数组来实现循环队列，且当前的rear和front分别为0和3，当执行两次出队操作，再执行两次入队操作，再执行一次出队操作后，rear和front的值分别是多少？
- 4、写出使用两个栈来模拟一个队列的伪代码。
- 5、元素a,b,c,d,e依次进入初始为空的栈中，若元素进栈后可停留，可出栈，直到所有元素都出栈，则在所有可能的出栈顺序中，以元素d开头的序列有几个？

栈的练习题目（此题不用交）复习做参考

为了充分利用空间，可将两个顺序栈存储在一个一维数组中。设计一个共享方式，画出示意图。然后定义一个实现双栈的类，这个类必须实现栈的所有标准操作，允许在栈的各类操作中增加一个指出对哪一个栈进行操作的参数。



Thanks! & QA