



Data Structures

栈和队列 Stacks & Queues

2022年9月21日

学而不厌 诲人不倦

- ➡ **3.1 引言**
- ➡ **3.2 栈**
- ➡ **3.3 队列**
- ➡ **3.4 扩展与提高**
- ➡ **3.5 应用举例**

3.3 队列

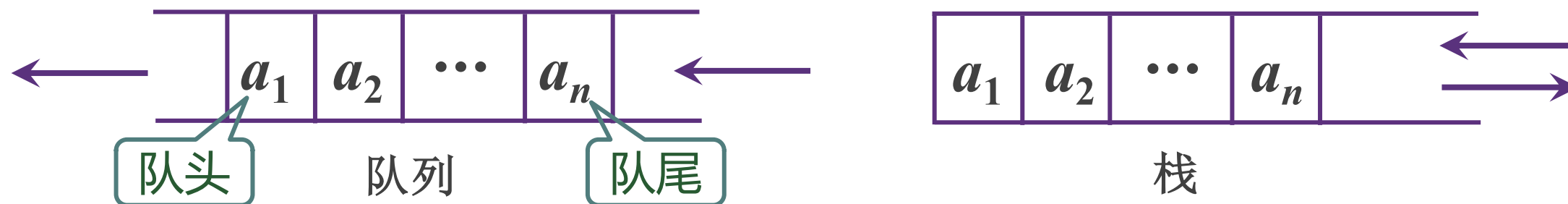
3-3-1 队列的逻辑结构



1. 队列的定义

✦ 队列：只允许在表的一端进行插入操作，在另一端进行删除操作

$$(a_1, a_2, \dots, a_{n-1}, a_n)$$



✦ 队尾：允许插入的一端，相应地，位于队尾的元素称为队尾元素

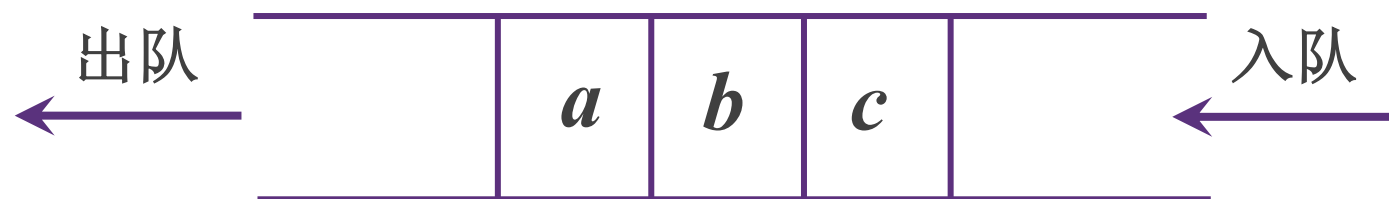
队头：允许删除的一端，相应地，位于队头的元素称为队头元素



2. 队列的操作特性

例：有三个元素按 a 、 b 、 c 的次序依次入队，且每个元素只允许进一次队，则出队序列是什么？

答：出队序列只有一种情况： $a\ b\ c$



插入：入队、进队

删除：出队

✦ 空队列：不含任何数据元素的队列

队列的操作特性：先进先出 (**F**irst **I**n **F**irst **O**ut , **FIFO**)



3. 队列的抽象数据类型定义

ADT Queue

DataModel

队列中元素具有相同类型及先进先出特性，相邻元素具有前驱和后继关系

Operation

InitQueue: 队列的初始化

DestroyQueue: 队列的销毁

EnQueue: 入队

DeQueue: 出队

GetQueue: 取队头元素

Empty: 判空

endADT

 与栈类似，队列的基本操作是确定的



3. 队列的抽象数据类型定义

InitQueue

输入：无

功能：初始化队列，创建一个空队列

输出：无

DestroyQueue

输入：无

功能：销毁队列，释放队列所占用的存储空间

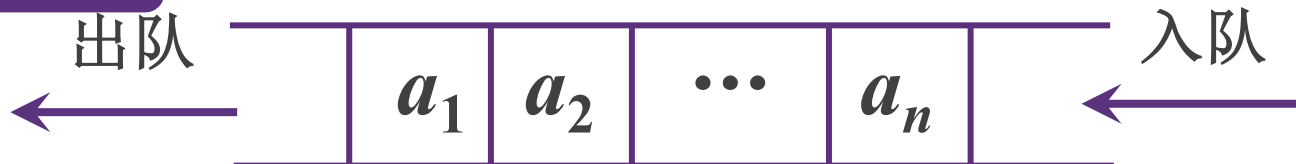
输出：无

EnQueue

输入：元素值 x

功能：在队尾插入一个元素

输出：如果插入成功，队尾增加了一个元素；否则返回失败信息



Enqueue操作需要指明插入位置吗？



3. 队列的抽象数据类型定义

DeQueue

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；否则给出失败信息

GetQueue

输入：无

功能：读取队头元素

输出：若队列不空，返回队头元素；否则给出失败信息

Empty

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



3.3 队列

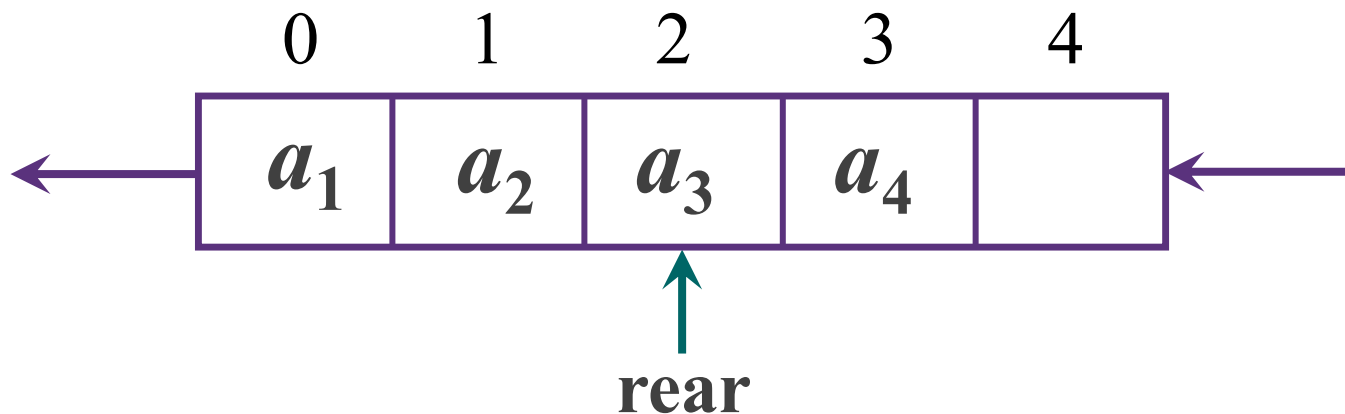
3-3-2 队列的顺序存储结构及实现



1. 顺序队列

✚ 顺序队列：队列的顺序存储结构

例： $a_1a_2a_3a_4$ 依次入队



🕒 如何改造数组实现队列的顺序存储？ 🕒 入队操作的时间性能？

✚ 如何表示队头：用数组的一端作为队头，从下标 0 处开始存放

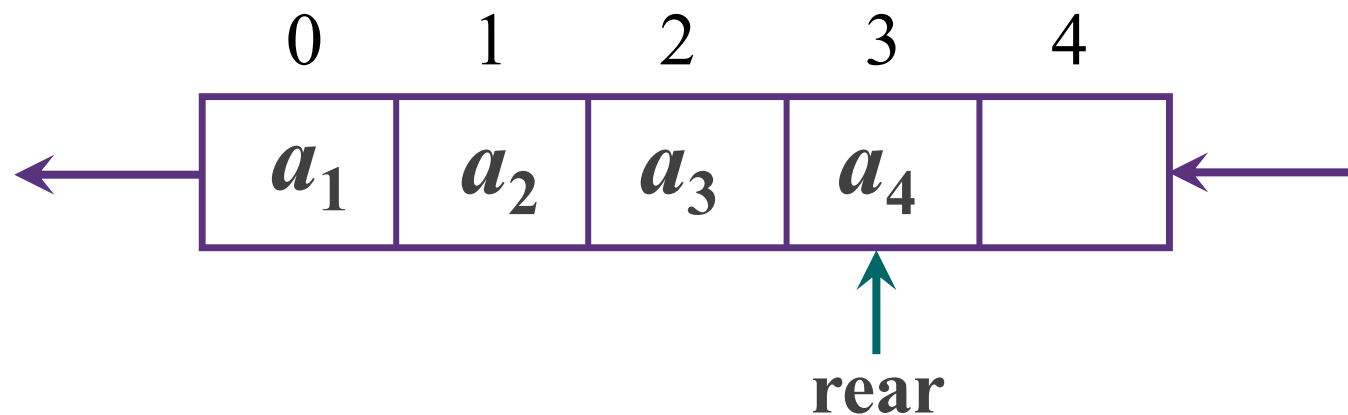
✚ 如何表示队尾：设变量rear存储队尾元素所在的下标



1. 顺序队列

✚ 顺序队列：队列的顺序存储结构

例： a_1a_2 依次出队



🕒 如何改造数组实现队列的顺序存储？

✚ 如何表示队头：用数组的一端作为队头，从下标 0 处开始存放

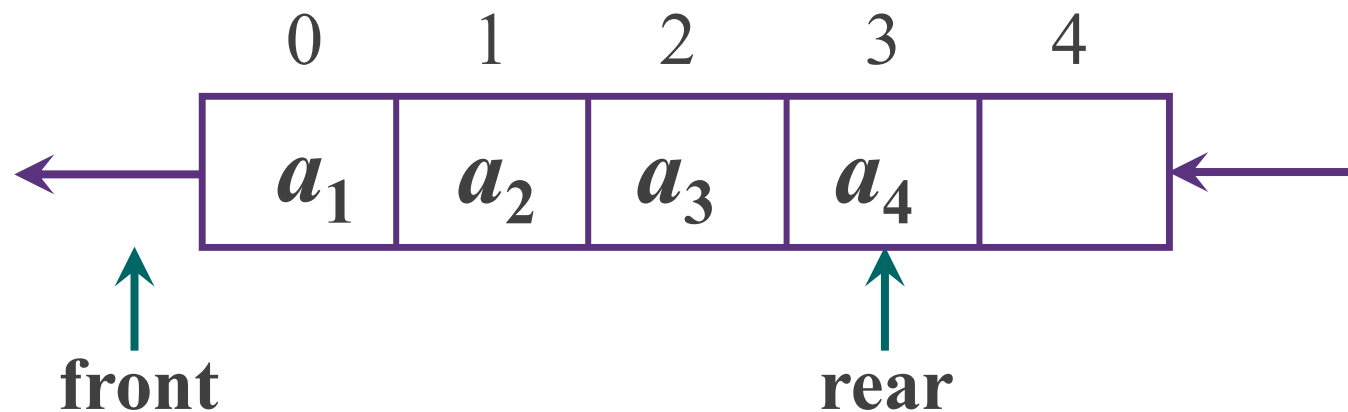
✚ 如何表示队尾：设变量rear存储队尾元素所在的下标



1. 顺序队列

 如何改进出队操作的时间性能？

例： a_1a_2 依次出队



✦ 设置队头、队尾两个位置指针 ✦ 入队、出队时间性能均是 $O(1)$

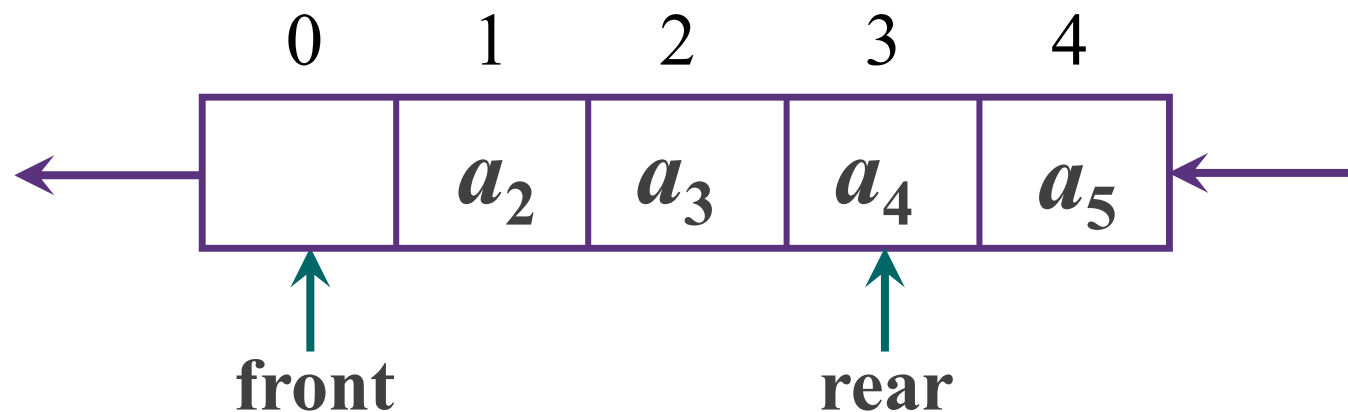
约定：队头front指向队头元素的前一个位置，队尾rear指向队尾元素



1. 顺序队列

 队列的移动有什么特点？

例： $a_1 a_2$ 依次出队



整个队列向数组下标较大方向移动



单向移动性

 队列的单向移动性会产生什么问题？

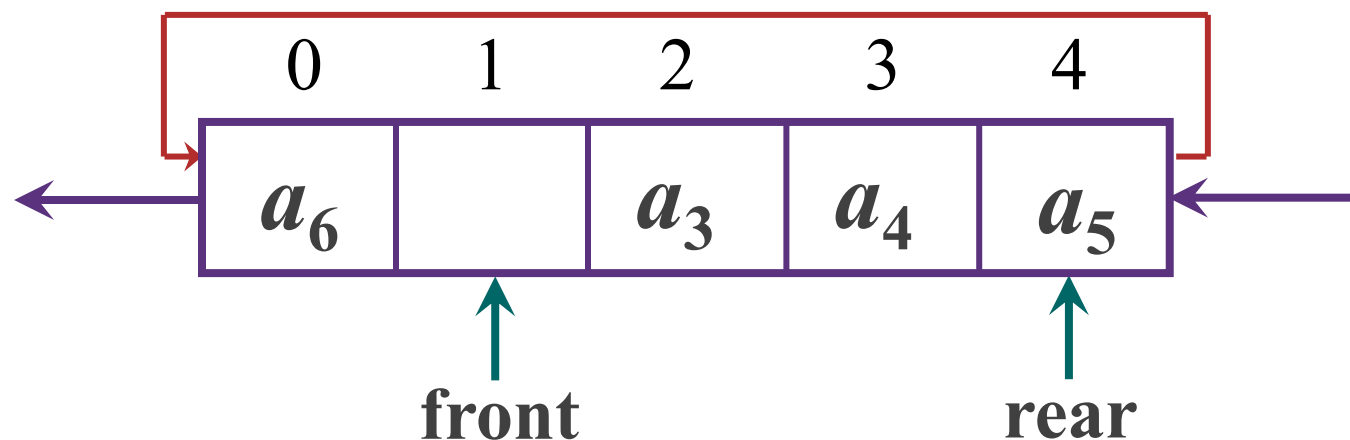
 **假溢出**：数组空间发生上溢，但数组的低端还有空闲空间



2. 循环队列

🕒 如何解决假溢出呢？

🕒 如何使数组下标循环呢？



📌 循环队列：队列采用顺序存储，并且数组是头尾相接的循环结构

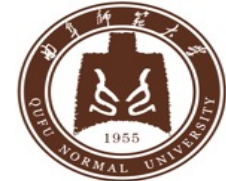
```
if (rear + 1 > 4)
    rear = 0;
else
    rear++;
```



$\text{rear} = (\text{rear} + 1) \% 5$

程序技巧：求模（正余数）使得数组下标循环

3.3 队列



3-3-2 队列的顺序存储结构及实现

3. 循环队列的类定义

 队列的抽象数据类型定义？

InitQueue: 队列的初始化

DestroyQueue: 队列的销毁

EnQueue: 入队

DeQueue: 出队

GetQueue: 取队头元素

Empty: 判空



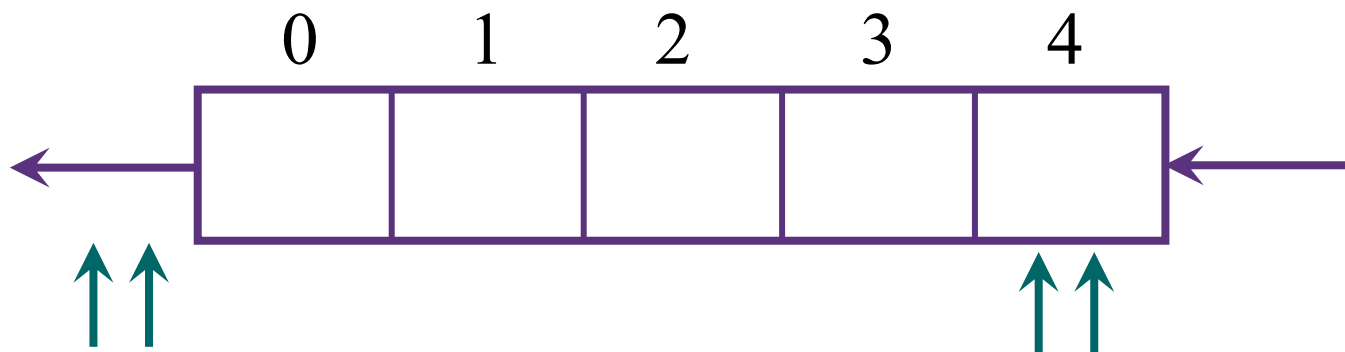
```
const int QueueSize = 100;
template <typename DataType>
class CirQueue
{
public:
    CirQueue( );
    ~CirQueue( );
    void EnQueue(DataType x);
    DataType DeQueue( );
    DataType GetQueue( );
    int Empty( );
private:
    DataType data[QueueSize];
    int front, rear;
};
```



4. 循环队列的实现——初始化

✦ 循环队列：队列采用顺序存储，并且数组是头尾相接的循环结构

✦ 设置队头、队尾两个位置指针



front rear

```
CirQueue<DataType> :: CirQueue( )  
{  
    front = rear = -1;  
}
```

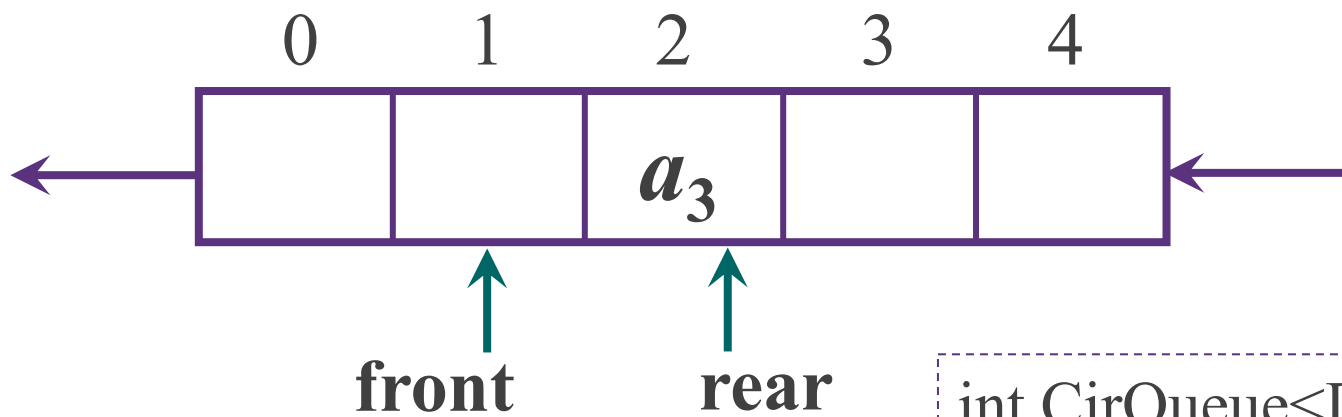
front rear

```
CirQueue<DataType> :: CirQueue( )  
{  
    front = rear = QueueSize - 1;  
}
```




4. 循环队列的实现——判空

🕒 如何判断循环队列的队空？

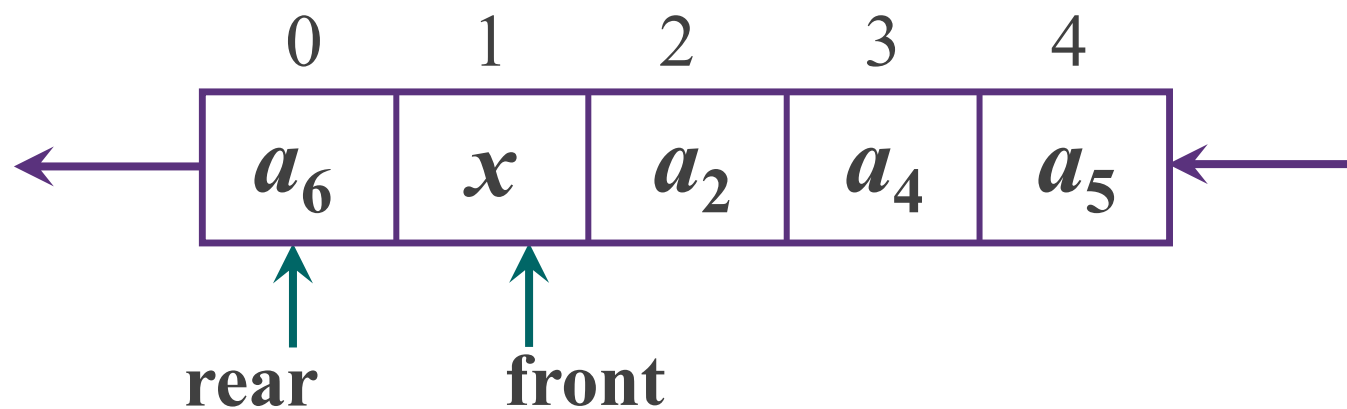


队空的判定条件： $\text{front} = \text{rear}$

```
int CirQueue<DataType> :: Empty( )
{
    if (rear == front) return 1;
    else return 0;
}
```

4. 循环队列的实现——队空/队满

 如何判断循环队列队满？



队空的判定条件： $\text{front} = \text{rear}$

队满的判定条件： $\text{front} = \text{rear}$

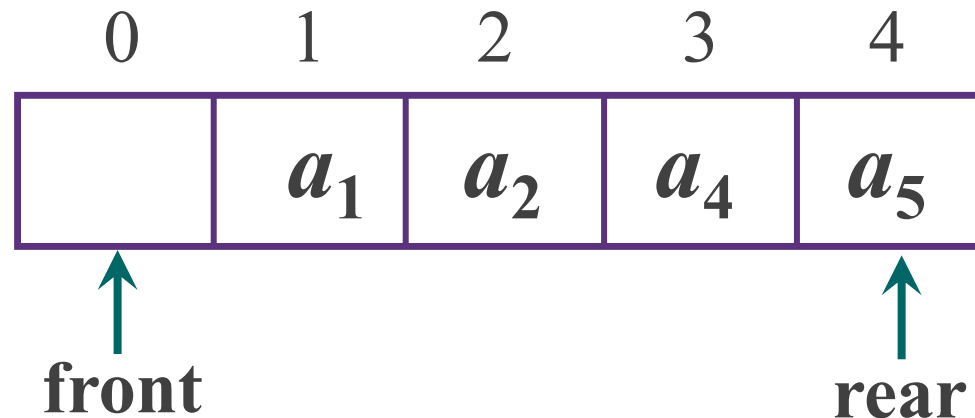
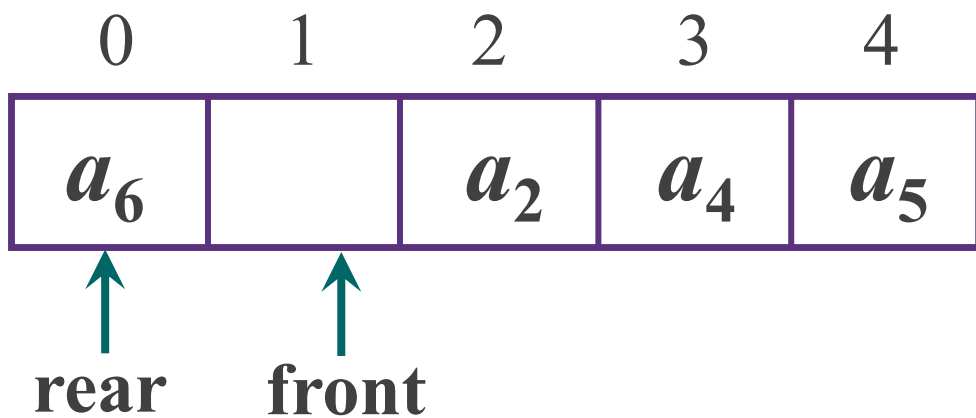


4. 循环队列的实现——队空/队满

队空的判定条件： $\text{front} = \text{rear}$

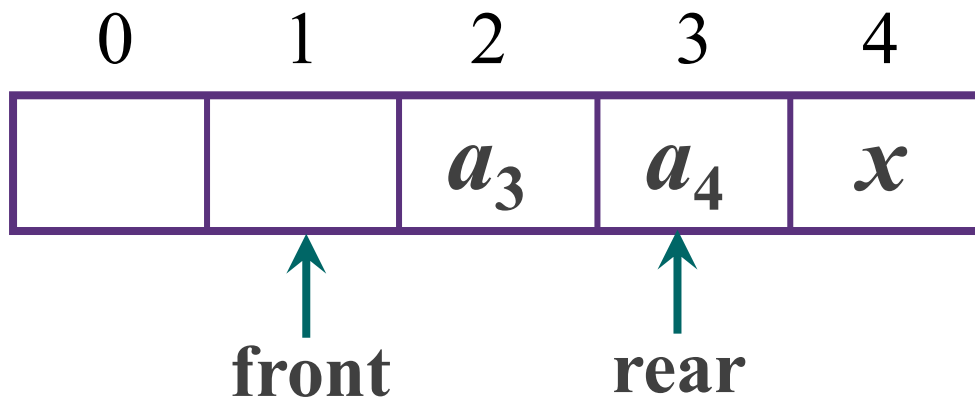
 队满的判定条件： \Rightarrow 数组中有一个空闲单元

$$(\text{rear} + 1) \% \text{QueueSize} = \text{front}$$





4. 循环队列的实现——入队

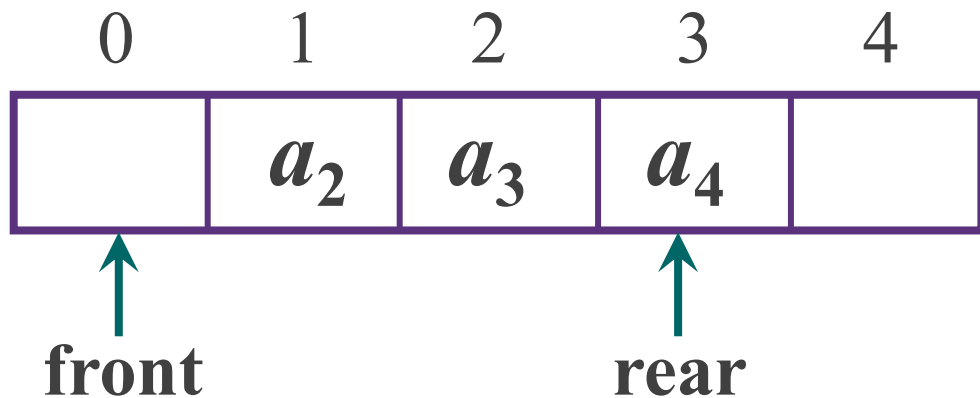


 时间复杂度是多少？

```
template <typename DataType>
void CirQueue<DataType> :: EnQueue(DataType x)
{
    if ((rear + 1) % QueueSize == front) throw "上溢";
    rear = (rear + 1) % QueueSize;           //队尾指针在循环意义下加1
    data[rear] = x;                          //在队尾处插入元素
}
```



4. 循环队列的实现——出队



 取队头元素的实现？

```
template <typename DataType>
DataType CirQueue<DataType> :: DeQueue( )
{
    if (rear == front) throw "下溢";
    front = (front + 1) % QueueSize;
    return data[front];
}
```

//队头指针在循环意义下加1
//读取并返回出队前的队头元素

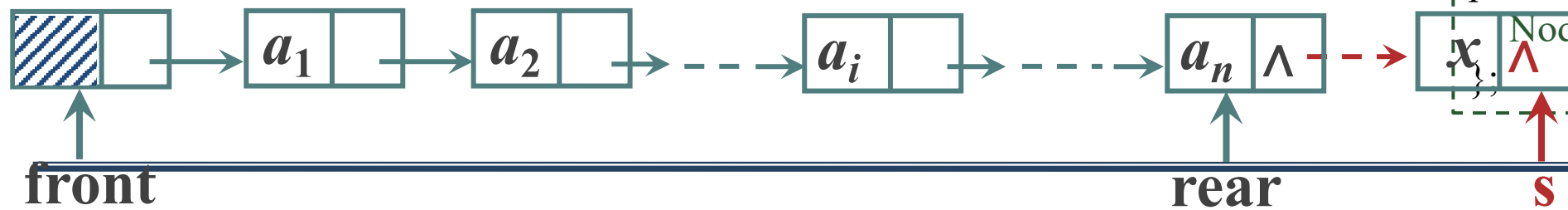
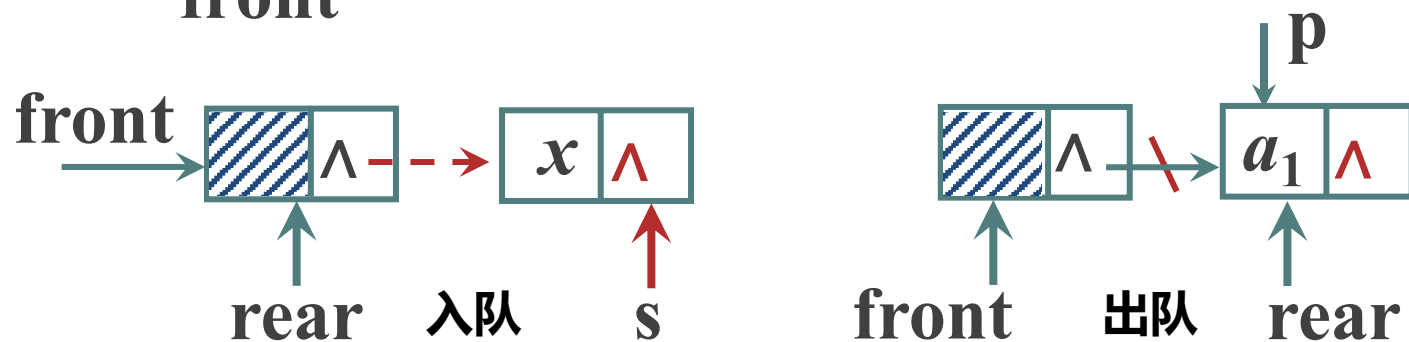
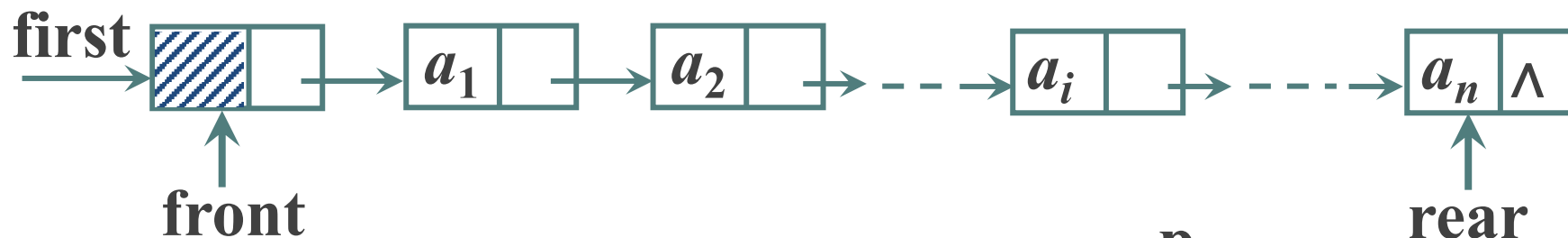
3.3 队列

3-3-3 队列的链接存储结构及实现



1. 链队列的存储结构定义

✦ 链队列：队列的链接存储结构



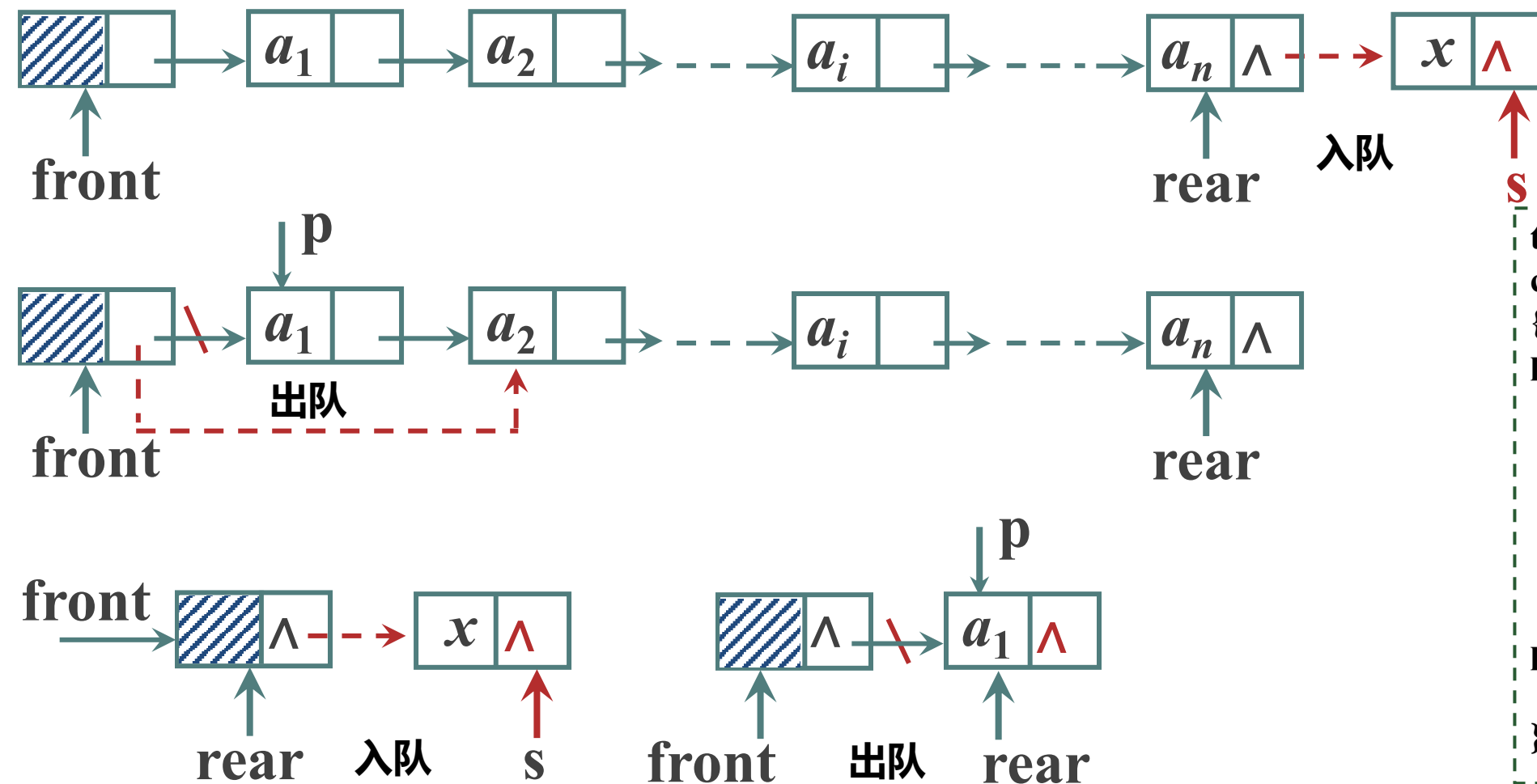
```
template <typename DataType>
struct Node
{
    DataType data;
    Node<DataType>*next;
};
```

```
template <typename DataType>
class LinkQueue
{
public:
    LinkQueue( );
    ~LinkQueue( );
    void EnQueue(DataType x);
    DataType DeQueue( );
    DataType GetQueue( );
    int Empty( );
private:
    Node<DataType> *front, *rear;
```



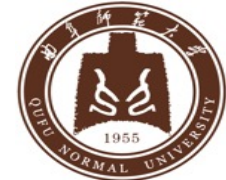
1. 链队列的存储结构定义

链队列：队列的链接存储结构



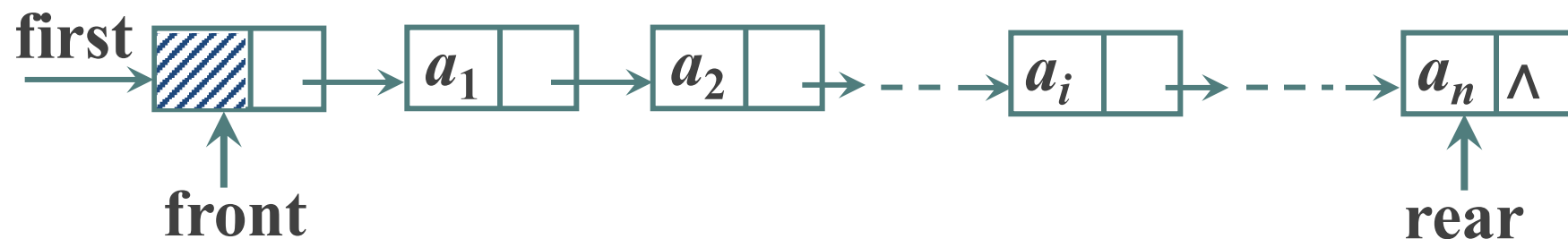
```
template <typename DataType>
struct Node
{
    DataType data;
    Node<DataType>*next;
};
```

```
template <typename DataType>
class LinkQueue
{
public:
    LinkQueue();
    ~LinkQueue();
    void EnQueue(DataType x);
    DataType DeQueue();
    DataType GetQueue();
    int Empty();
private:
    Node<DataType> *front, *rear;
};
```

1. 链队列的存储结构定义

✚ 链队列：队列的链接存储结构



🕒 P: 用链表的哪一端作为队头？哪一端作为队尾？

A: 链头作为队头，出队时间为 $O(1)$

链尾作为队尾，入队时间为 $O(n)$ \Rightarrow 设置队尾指针rear

🕒 P: 链队列需要加头结点吗？



2. 链队列的类定义

```
template <typename DataType>
struct Node
{
    DataType data;
    Node<DataType>*next;
};
```

InitQueue: 队列的初始化

DestroyQueue: 队列的销毁

EnQueue: 入队

DeQueue: 出队

GetQueue: 取队头元素

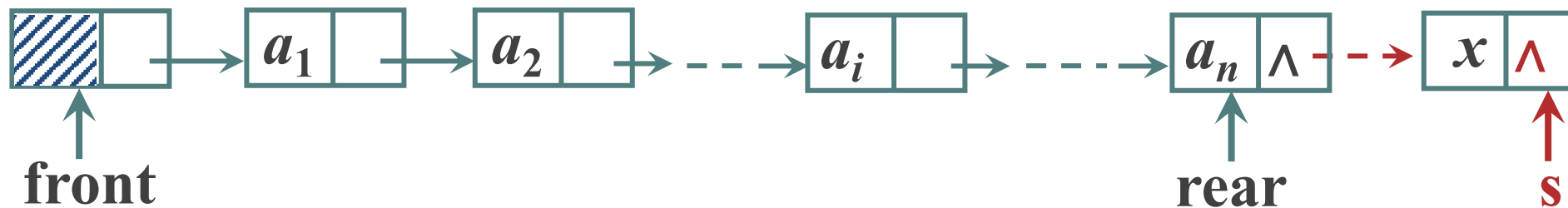
Empty: 判空



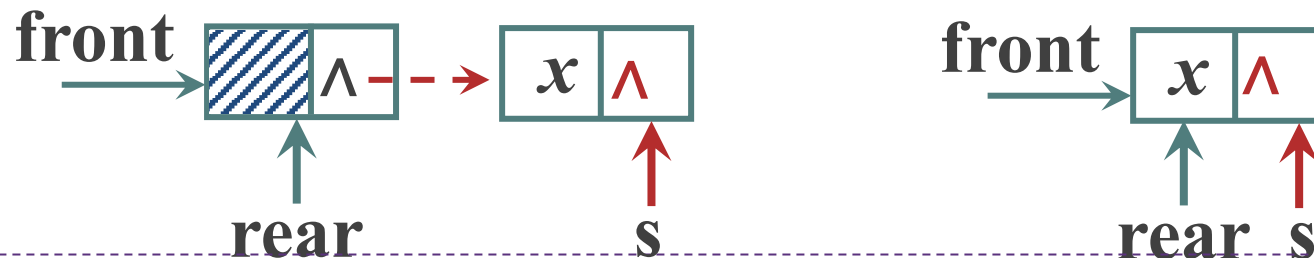
```
template <typename DataType>
class LinkQueue
{
public:
    LinkQueue( );
    ~LinkQueue( );
    void EnQueue(DataType x);
    DataType DeQueue( );
    DataType GetQueue( );
    int Empty( );
private:
    Node<DataType> *front, *rear;
};
```



3. 链队列的实现——入队



⌚ 没有头结点会怎样？



```
void LinkQueue<DataType> :: EnQueue(DataType x)
```

⌚ 时间复杂度是多少？

```
{
    Node<DataType> *s = nullptr;
    s = new Node<DataType>;
    s->data = x; s->next = nullptr;
    rear->next = s; rear = s;
```

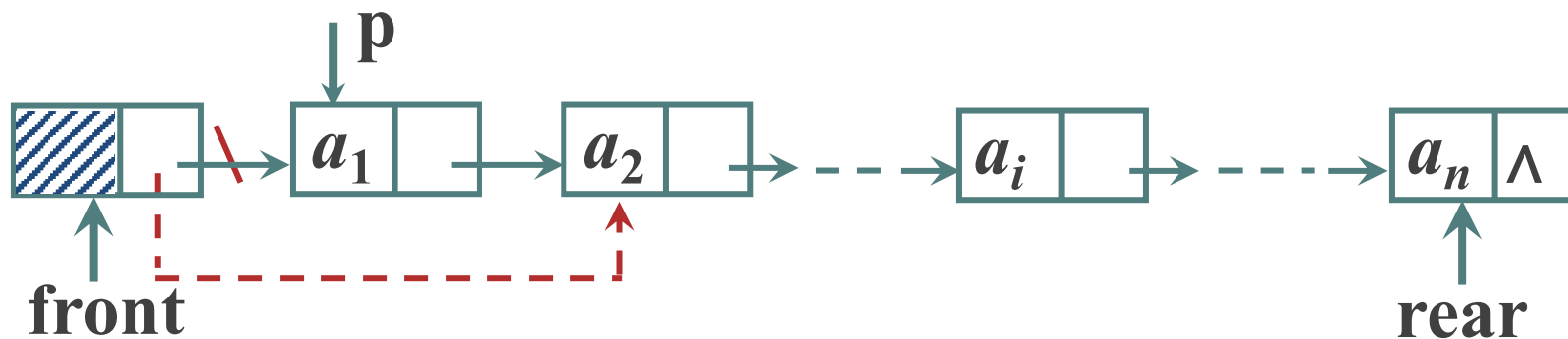
//申请结点s

rear = s; front = s;

```
}
```

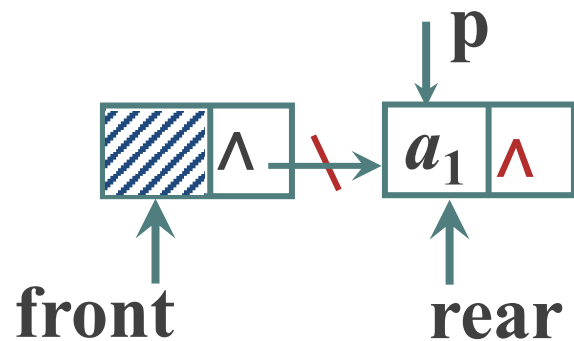


3. 链队列的实现——出队



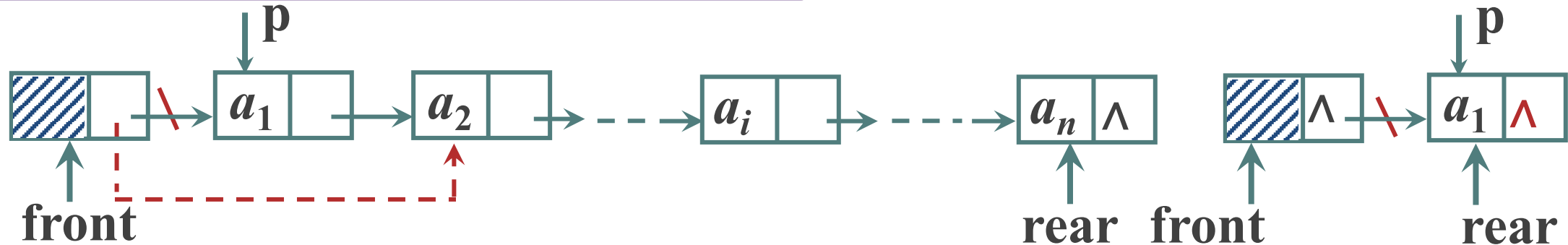
🕒 如何判断边界情况？

```
if (p->next == nullptr)
    rear = front;
```



🕒 考虑边界情况：队列中只有一个元素？

3. 链队列的实现——出队



```
DataType LinkQueue<DataType> :: DeQueue( )
```

```
{
```

```
    DataType x;
```

```
    Node<DataType> *p = nullptr;
```

```
    if (rear == front) throw "下溢";
```

```
    p = front->next; x = p->data;
```

```
    front->next = p->next;
```

```
    if (p->next == nullptr) rear = front;
```

```
    delete p;
```

```
    return x;
```

```
}
```



取队头元素的实现？

小结

1. 栈、队列的定义、结构特点
2. 栈、队列的顺序存储、链式存储
3. 栈、队列是一种特殊（操作受限）的线性表。 栈、队列的独特操作
4. 栈空、栈满、队空、队满的判定条件
5. 栈的应用： 利用栈可将队列的所有元素逆置

一般线性表

逻辑结构：一对一

存储结构：顺序表、链表

运算规则：随机、顺序存取

栈

逻辑结构：一对一

存储结构：顺序栈、链栈

运算规则：后进先出

队列

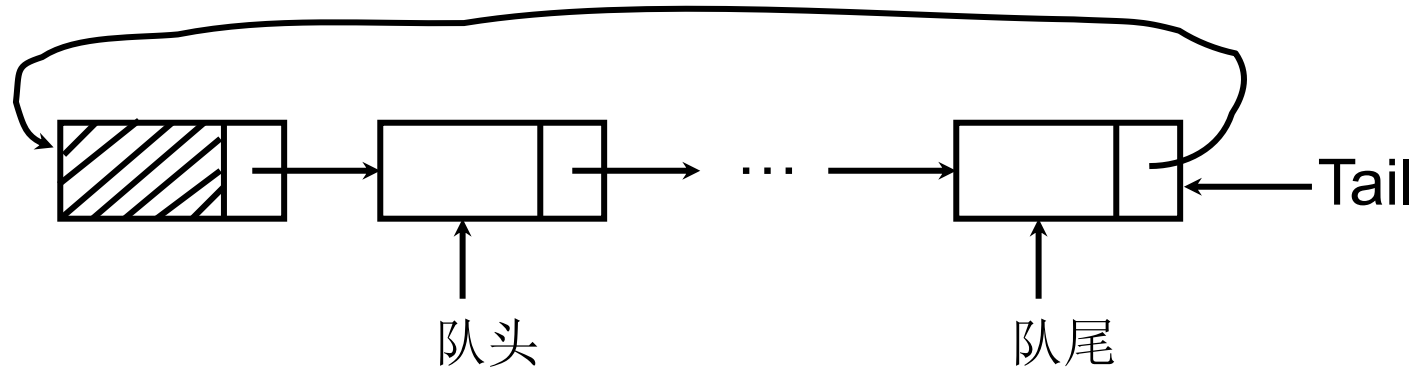
逻辑结构：一对一

存储结构：顺序队、链队

运算规则：先进先出

作业

1. 总结栈空、栈满、队空、队满的判定条件。
2. 循环队列的优点是什么？设用数组来存放循环队列，你有几种判断队满和队空的方案？
3. 假设以带头结点的循环单链表表示队列，并且只设一个尾指针，试编写入队和出队算法。



实验三、栈和队列的应用

一、实验目的

1. 掌握顺序栈和链式栈的定义与操作方法。
2. 掌握循环队列和链式队列的定义与操作方法。
3. 掌握栈和队列的空、满状态判断方法。
4. 用C++语言实现顺序栈、链式栈、循环队列和链式队列，并上机调试。

二、实验内容

1. 设计顺序栈SeqStack和链式栈LinkStack及相关方法，用于维护栈的基本操作。
2. 设计循环队列CirQueue和链式队列LinkQueue及相关方法，用于维护队列的基本操作。
3. 设计算法实现简单表达式求值、十进制转换为二至九进制之间的任一进制。（**任选一个**）
4. 给出测试过程和测试结果。

实验时间： 第6周周四晚 （10月7日补）
22网安： 18:30-20:10 **22物联网：** 20:10-21:50
实验地点： 软件基础实验室301（老干部处）



Thank You !

Q & A