第三章 栈和队列

机电工程与自动化学院 L栋301

任卫红 助理教授

renweihong@hit.edu.cn

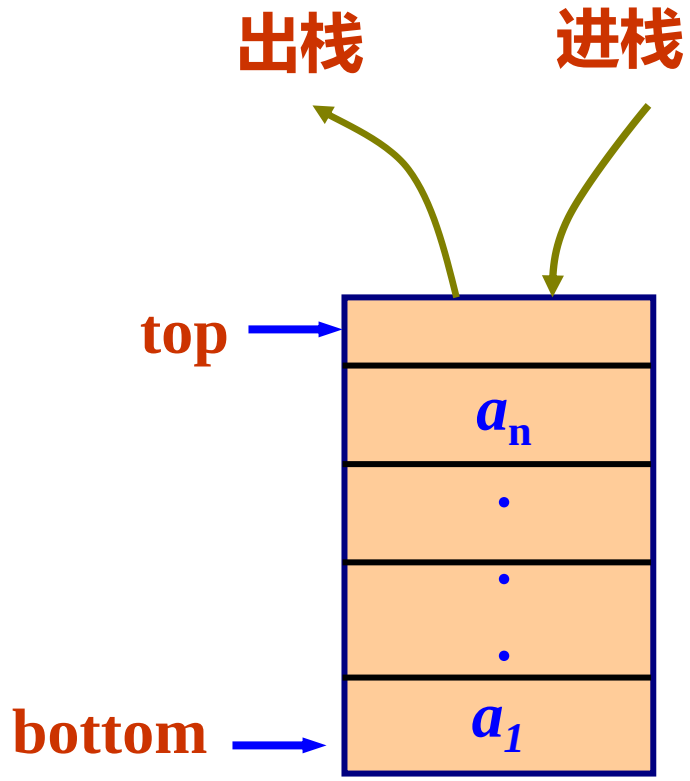
http://faculty.hitsz.edu.cn/renweihong

3.1栈

第一节　栈

一、栈

- 栈是限定仅在表尾(top)进行插入或删除操作的线性表。
- 允许插入和删除的一端称为栈顶(top，表尾)，另一端称为栈底(bottom，表头)
- 特点：后进先出 (LIFO)

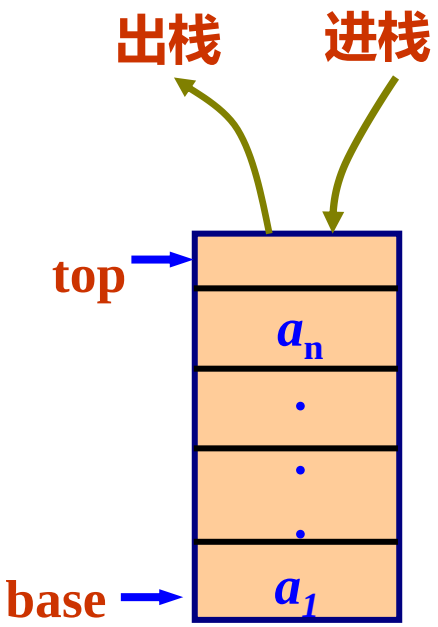


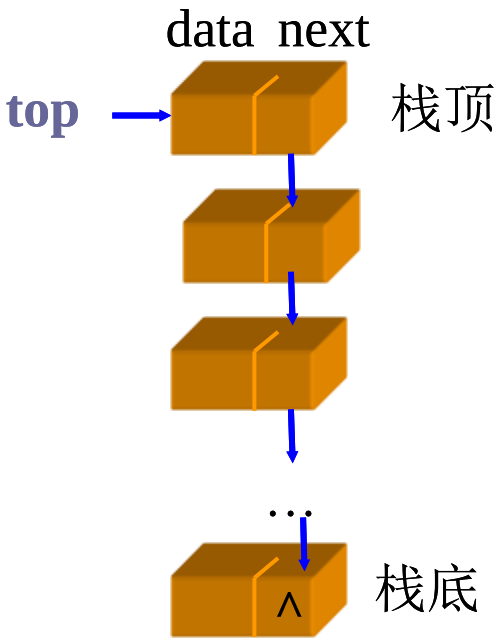
第一节　栈

二、栈的实现

- 栈的存储结构主要有两种：

1. 顺序栈
2. 链式栈



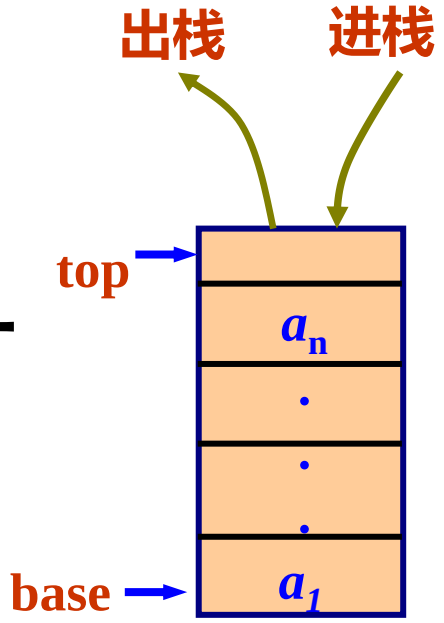


3.2 顺序栈

第二节　顺序栈

一、顺序栈

- 顺序栈是栈的顺序存储结构
- 利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素
- 指针top指向栈顶元素在顺序栈中的下一个位置，
- base为栈底指针，指向栈底的位置。



第二节　顺表栈

二、顺序栈的定义

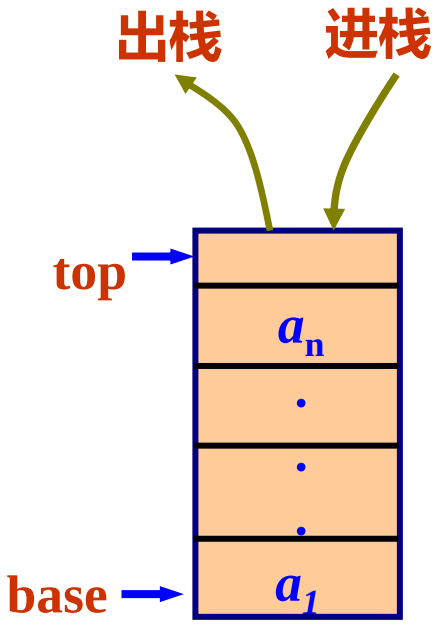
- 采用动态分配的一维数组表示顺序表

```
template <class T>
class SStack
{
    T           *base;      //顺序栈基址
    int         top;        //ElemType *top;
    int         MaxSize;    //栈容量
public:
    SStack();               //初始化
    void Push(T e);         //入栈
    T Pop();                //出栈
    bool IsFull();          //判栈满。满，返回true;否则，返回false
    bool IsEmpty();         //判栈空。空，返回true;否则，返回false
    T GetTop();             //得到栈顶元素，不出栈
    ~SStack();              //析构，释放空间
};
```

第二节　顺序栈

三、顺序栈的特性

- top=0或top=base表示栈空
- base=NULL表示栈不存在
- 当插入新的栈顶元素时，指针top+1
- 删除栈顶元素时，指针top-1
- 当top>stacksize时，栈满，溢出



第二节　顺序栈

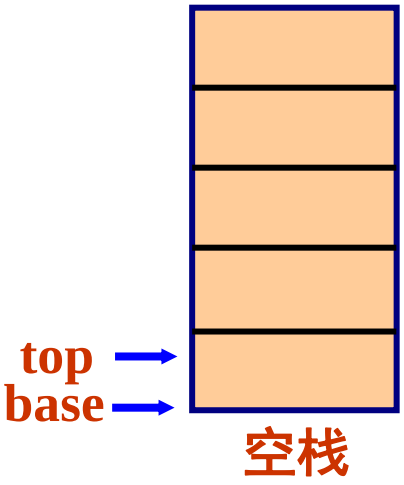
四、顺序栈的主要操作

```
template <class T>
class SStack
{
    T           *base;      //顺序栈基址
    int         top;        //ElemType *top;
    int         MaxSize;    //栈容量
public:
    SStack();               //初始化
    void Push(T e);         //入栈
    T Pop();                //出栈
    bool IsFull();          //判栈满。满，返回true;否则，返回false
    bool IsEmpty();         //判栈空。空，返回true;否则，返回false
    T GetTop();             //得到栈顶元素，不出栈
    ~SStack();              //析构，释放空间
};
```

第二节　顺序栈

五、创建顺序栈

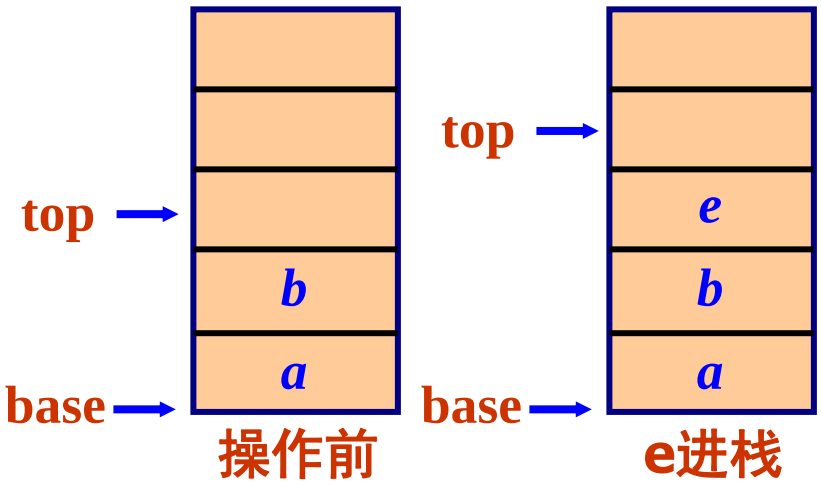
```
template <class T>
SStack<T>::SStack()    //创建栈
{
    MaxSize = 100;
    base = new T[MaxSize];
    if(!base)
    {
        cout<<"malloc error!"<<endl;
        exit(-1);
    }
    top = 0;           //空栈
}
```



第二节　顺序栈

六、进栈（插入新元素）

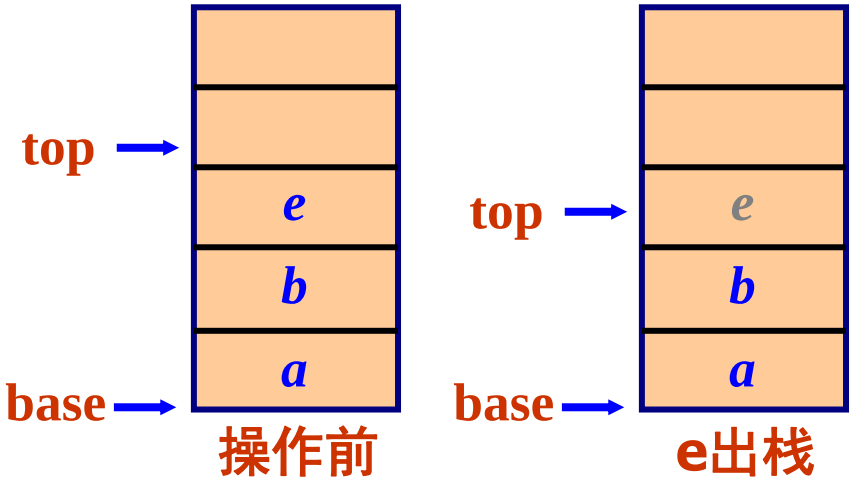
```
template <class T>
void SStack<T>::Push(T e)
{
    base[top++] = e;
}
```



第二节　顺序栈

七、出栈（删除元素）

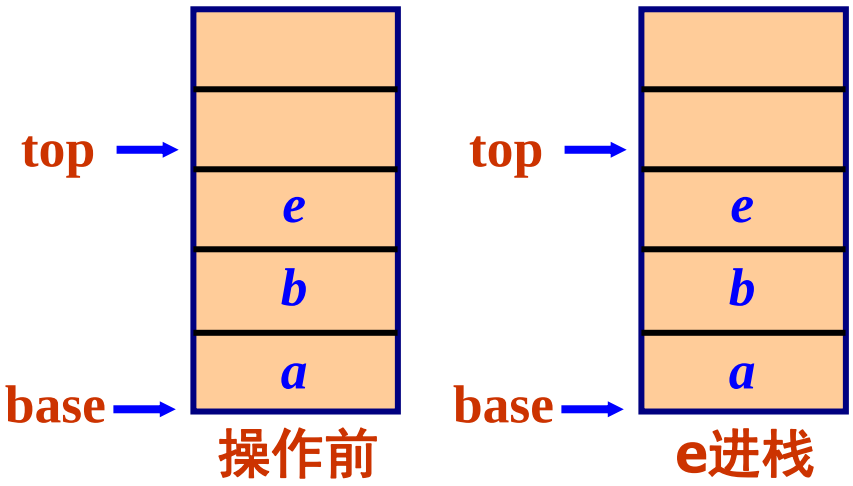
```
template <class T>
T SStack<T>::Pop()
{
    return base[--top];
}
```



第二节　顺序栈

八、判栈满

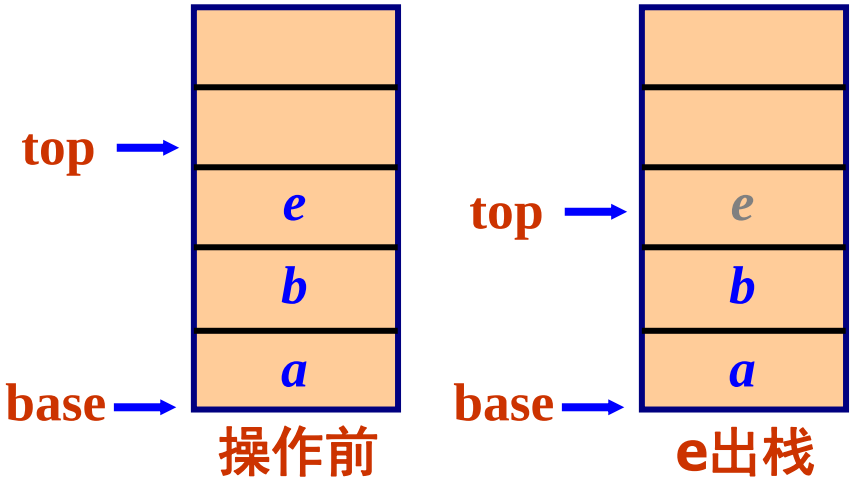
```
template <class T>
bool SStack<T>::IsFull()
{
    if(top == MaxSize)
        return true;
    return false;
}
```



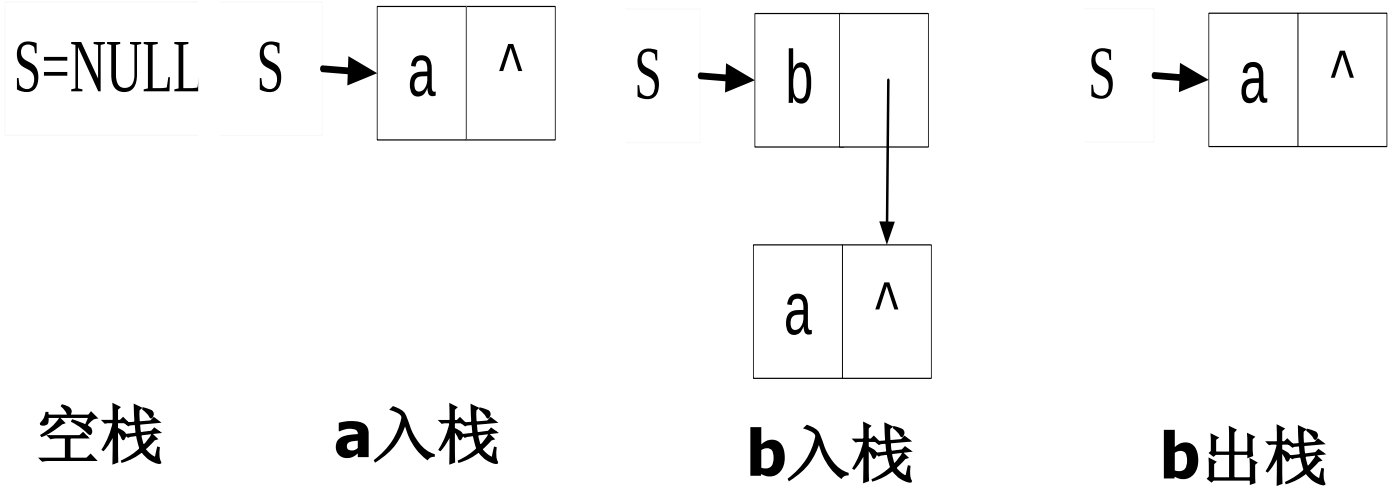
第二节　顺序栈

九、其它操作

- 判栈空
- 获得栈顶元素



链栈



空栈　　a入栈　　b入栈　　b出栈

链栈在栈底（表头）进行入栈，出栈操作。

链栈结构定义

栈的链式存储结构称为链栈，它是运算受限的单链表，插入和删除操作仅限制在表头位置上进行。

由于只能在链表头部进行操作，故链表没有必要像单链表那样附加头结点。栈顶指针就是链表的头指针。

C++中栈容器

stack 模板类的定义在<stack>头文件中。

定义stack 对象的示例代码如下：

```
stack<int> s1;
stack<string> s2;
```

stack 的基本操作有：

入栈，如例：s.push(x);

出栈，如例：s.pop();注意，出栈操作只是删除栈顶元素，并不返回该元素。

访问栈顶，如例：s.top()

判断栈空，如例：s.empty()，当栈空时，返回true。

访问栈中的元素个数，如例：s.size()。

3.3 栈的应用举例

第三节　栈的应用举例

一、数值转换(八进制)

- 将十进制转换为其它进制(d)，其原理为：

$$N = (N/d)*d + N \bmod d$$

例如：(1348)10 = (2504)8 ，其运算过程如下：

计算顺序 ↓

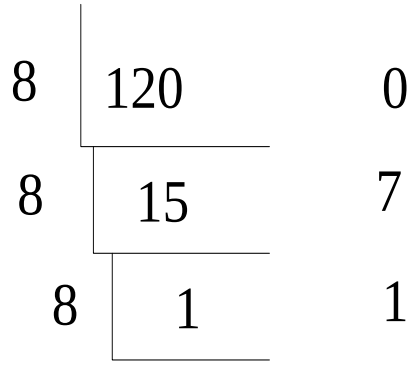
N	N /8	N mod 8
1348	168	4
168	21	0
21	2	5
2	0	2

↑ 输出顺序

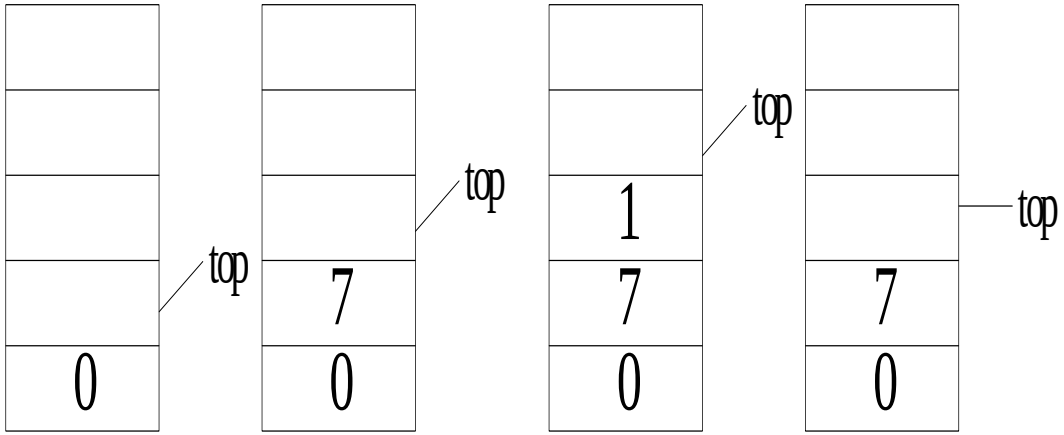
- 数制转换

10进制到d (2,8,16)的转换公式：

$$(N)_{10} = (N / d) * d + N \bmod d$$



$(120)_{10} = (170)_8$



第三节　栈的应用举例

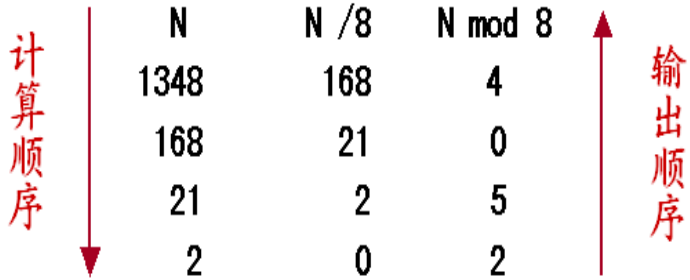
一、数值转换(八进制)

```
#include <stack>          //C++中已定义栈容器
#include <iostream>
using namespace std;

void  Converse(int n)
 {
     stack<int>  s;        //创建栈

     do{
         s.push(n%8);     //入栈
         n/=8;
     }while(n);

     while(!s.empty())    //栈非空
     {
         printf("%d",s.top());    //输出栈顶元素
         s.pop();                 //出栈
     }
 }
```



第三节　栈的应用举例

二、行编辑程序

- 用户输入一行字符
- 允许用户输入出差错，并在发现有误时，可以用退格符“#”及时更正
- 假设从终端接受两行字符：

  whli##ilr#e（s#*s)
- 实际有效行为：

  while (*s)

第三节　栈的应用举例

二、行编辑程序

- 对用户输入的一行字符进行处理，直到行结束（“\n”）

```
string   str;
stack<char> s,s1;          //创建栈
int      i;

getline(cin,str);          //输入一行字符串，允许空格

for(i<0; i<str.length();i++)
    if(str[i]!='#')
        s.push(str[i]);
    else
        s.pop();

//将s栈字符从栈顶至栈底拷贝至栈s1中
//输出s1栈字符内容
```

第三节　栈的应用举例

三、迷宫求解

- 迷宫求解一般采用“穷举法”
- 逐一沿顺时针方向查找相邻块（一共四块－东(右)、南(下)，西(左)、北(上)）是否可通，即该相邻块既是通道块，且不在当前路径上
- 用一个栈来记录已走过的路径

第三节　栈的应用举例

三、迷宫求解

- 举例

#	#	#	#	#	#	#	#
#	★		#				#
#		#	#			#	#
#		#					#
#		#		#	#	#	#
#						#	#
#		#				○	#
#	#	#	#	#	#	#	#

第三节　栈的应用举例

三、迷宫求解（算法）

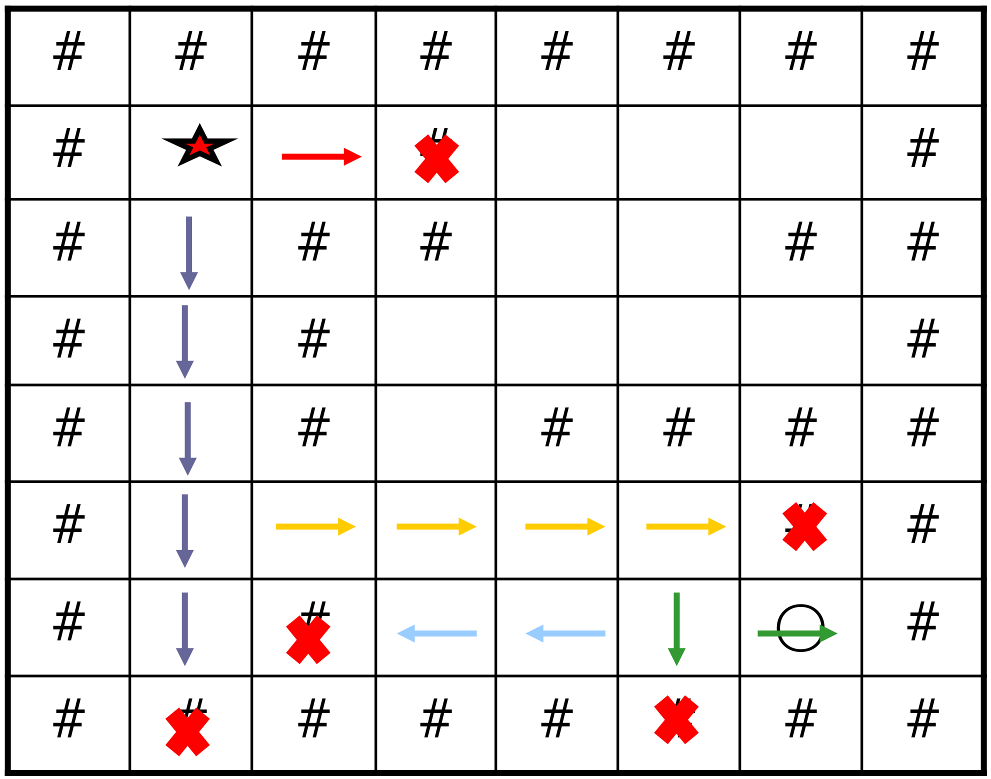
- 设定当前位置为入口位置

do {若当前位置可通，则 {
将该位置插入栈顶(Push)；
若该位置是出口，则结束；
否则切换当前位置的东邻方块为当前位置；
}
否则 {若栈不空且栈顶位置尚有其他位置未搜索则{
设定新的当前位置为顺时针方向旋转找到下一邻块}
若栈不空但栈顶位置四周均不可通则
{删去栈顶位置；
若栈不空，则重新测试新的栈顶位置，
只到找到一个可通的相邻块或出栈至栈空；}
}
}while(栈不空)

第三节　栈的应用举例

三、迷宫求解

- **举例**



第三节　栈的应用举例

四、表达式求值

表达式由操作数、运算符和界限符组成，它们皆称为单词

- 操作数：常数或变量
- 运算符：+，－，*，／ 等
- 界限符：（，），#(表达式开始及结束符)

第三节　栈的应用举例

四、表达式求值

思考：表达式12+3*5+（2+10）*5 的计算顺序。

计算步骤：假设操作数栈NS和运算符栈OS，

（1）依次读取表达式，若为操作数，则直接进栈；

若为运算符(记为op2)，转（2）

（2）将op2与运算符栈顶元素(记为op1)按P53的表3.1比较优先权，并按如下规则进行操作：

- 若prec(op1) < prec(op2), 则op2入OS;
- 若prec(op1) = prec(op2), 则op1出栈，回到（1）；
- 若prec(op1) > prec(op2), 则NS出2个操作数num2,num1,op1出栈，计算num2 op1 num2, 结果入NS；回到(2) 。

（3）重复（1）、（2）直至整个表达式求值完毕。

- 例：计算表达式12+（2+10）*5 对应的栈变化如下图。

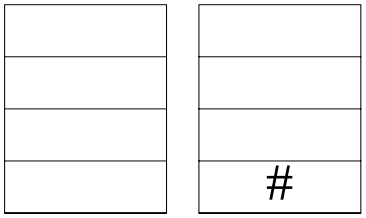
增加了一个输入结束符’#’，人为的在OS栈也加入了栈底’ #”。

考虑：如何提取操作数？

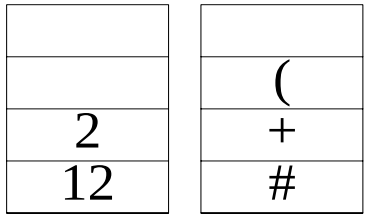
如何实现优先级比较？

num1 op1 num2如何实现？

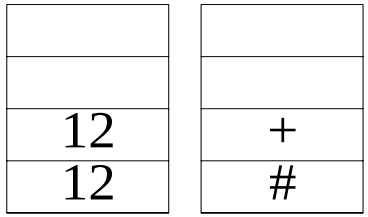
12+（2+10）*5



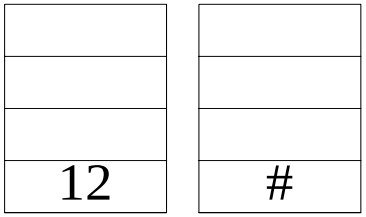
初始



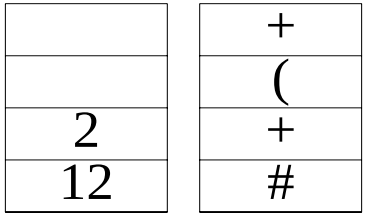
2



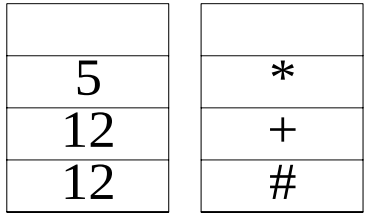
),(=)，(出



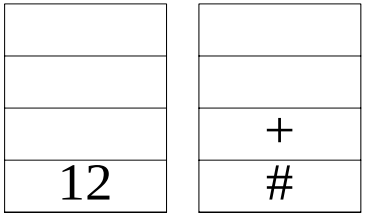
12



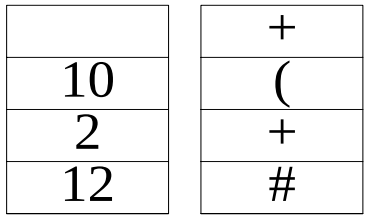
+, (< +



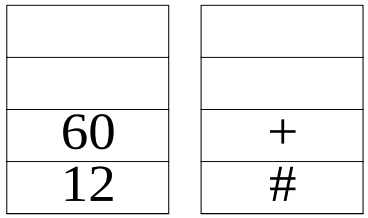
5，*，+<*



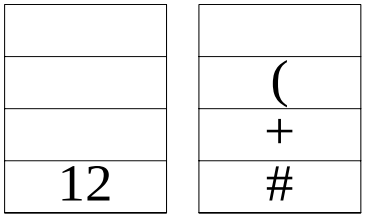
+, # < +



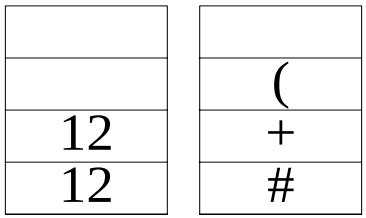
10



#，*>#, 计算



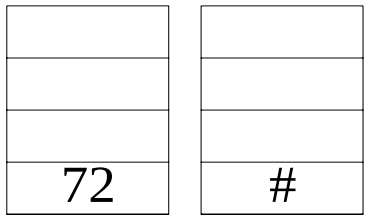
(, + < (



), + >), 2,10,+

出栈，写入

2+10



栈顶=#，结束

栈的应用：表达式的计算

补充

- 算术表达式有三种表示：
  - 中缀(infix)表示

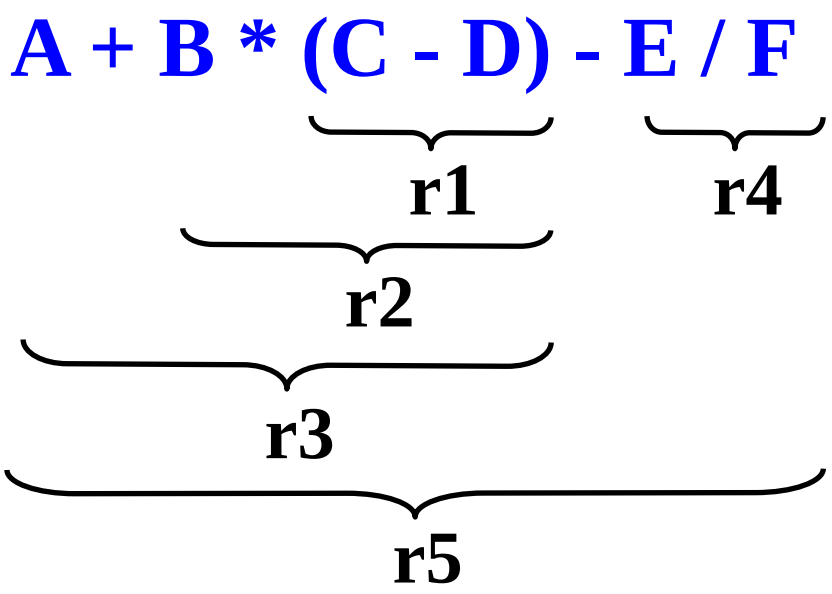
    <操作数> <操作符> <操作数>，如 A+B；
  - 前缀(prefix)表示

    <操作符> <操作数> <操作数>，如 +AB；
  - 后缀(postfix)表示

    <操作数> <操作数> <操作符>，如 AB+；

表达式示例

补充



中缀表达式 → 后缀表达式

A + B * (C - D) - E / F　　ABCD-*+EF/-

中缀表达式　A + B * (C - D) - E / F

补充

- 表达式中相邻两个操作符的计算次序为：
  - 优先级高的先计算
  - 优先级相同的自左向右计算
  - 当使用括号时从最内层括号开始计算

后缀表达式　A B C D - * + E F / -

- 在后缀表达式的计算顺序中已隐含了加括号的优先次序，括号在后缀表达式中不出现。

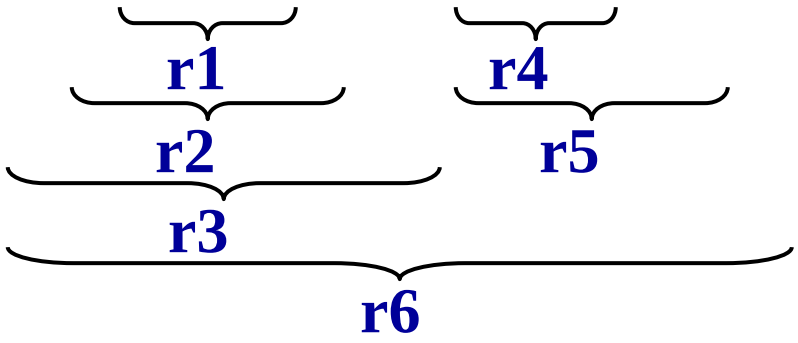
应用后缀表示计算表达式的值 补充

Idea：

- 从左向右顺序地扫描表达式，并用一个栈暂存扫描到的操作数或计算结果。
- 扫描中遇操作数则压栈；遇操作符则从栈中退出两个操作数，计算后将结果压入栈
- 最后计算结果在栈顶

补充

- 计算例 A B C D - * + E F ^ G / -



计算后缀表达式 A B C D - * + E F ^ G / -

步	输入	类 型	动 作	栈内容
1			置空栈	空
2	A	操作数	进栈	A
3	B	操作数	进栈	AB
4	C	操作数	进栈	ABC
5	D	操作数	进栈	ABCD
6	–	操作符	D、C 退栈，计算 C–D，结果 r1 进栈	ABr1
7	*	操作符	r1、B 退栈，计算 B*r1，结果 r2 进栈	Ar2
8	+	操作符	r2、A 退栈，计算 A+r2，结果 r3 进栈	r3

计算后缀表达式 **A B C D - * + E F ^ G / -**

步	输入	类 型	动 作	栈内容
9	E	操作数	进栈	r3E
10	F	操作数	进栈	r3EF
11	^	操作符	F、E 退栈，计算 E^F，结果 r4 进栈	r3r4
12	G	操作数	进栈	r3r4G
13	/	操作符	G、r4 退栈，计算 r4/G，结果 r5 进栈	r3r5
14	−	操作符	r5、r3 退栈，计算 r3−r5，结果 r6 进栈	r6

- **一般表达式的操作符有4种类型：**

补充

算术操作符 如双目操作符（+、–、*、/ 和%）以及单目操作符（-）。

关系操作符 包括<、<=、==、!=、>=、>。这些操作符主要用于比较。

逻辑操作符 如与(&&)、或(||)、非(!)。

括号 ‘(’和 ‘)’，它们的作用是改变运算顺序。

利用栈将中缀表示转换为后缀表示[了解]

- **各个算术操作符的优先级**

操作符 ch	;	(	*, /, %	+, −	)
isp (栈内)	0	1	5	3	6
icp (栈外)	0	6	4	2	1

- isp叫做栈内(in stack priority)优先级
- icp叫做栈外(in coming priority)优先级。
- 操作符优先级相等的情况只出现在括号配对或栈底的“#”号与输入流最后的“#”号配对时。

利用栈将中缀表示转换为后缀表示

- **转换步骤**
  - 操作符栈初始化，将结束符 ‘#’进栈。然后读入中缀表达式字符流的首字符ch。
  - 重复执行以下步骤，直到ch = ‘#’，同时栈顶的操作符也是 ‘#’，停止循环。
    - 若ch是操作数直接输出，读入下一个字符ch。
    - 若ch是操作符，判断ch的优先级icp和位于栈顶的操作符op的优先级isp：

利用栈将中缀表示转换为后缀表示

- **转换步骤**
  - 若 icp(ch) > isp(op)，令ch进栈，读入下一个字符ch。
  - 若 icp(ch) < isp(op)，退栈并输出。
  - 若 icp(ch) == isp(op)，退栈但不输出，若退出的是“(”号读入下一个字符ch。
- 算法结束，输出序列即为所需的后缀表达式

A+B*(C-D)-E/F

步	输入	栈内容	语义	输出	动作
1		#			栈初始化
2	A	#		A	操作数A输出，读字符
3	+	#	+ > #		操作符+进栈，读字符
4	B	#+		B	操作数B输出，读字符
5	*	#+	* > +		操作符*进栈，读字符
6	(	#+*	(> *		操作符(进栈，读字符
7	C	# +*(		C	操作数C输出，读字符
8	-	# +*(	- > (		操作符-进栈，读字符
9	D	# +*(-		D	操作数D输出，读字符
10	)	# +*(-	) < -	–	操作符–退栈输出
11		# +*(	) = (		(退栈，消括号，读字符

A+B*(C-D)-E/F

步	输入	栈内容	语义	输出	动作
12	-	#+*	- < *	*	操作符*退栈输出
13		#+	- < +	+	操作符+退栈输出
14		#	- > #		操作符-进栈，读字符
15	E	#-		E	操作数E输出，读字符
16	/	#-	/ > -		操作符/进栈，读字符
17	F	#-/		F	操作数F输出，读字符
18	#	#-/	# < /	/	操作符/退栈输出
19		#-	#< −	−	操作符−退栈输出
20		#	# = #		#配对，转换结束

五、栈与递归

- **递归的定义**

  若一个对象部分地包含它自己，或用它自己给自己定义，则称这个对象是递归的；若一个过程直接地或间接地调用自己，则称这个过程是递归的过程。

- 以下三种情况常常用到递归方法。
  - 定义是递归的
  - 数据结构是递归的
  - 问题的解法是递归的

五、栈与递归

例1：阶乘函数

$$n!=\begin{cases}1, & \text{当}n=0\text{时}\\ n*(n-1)!, & \text{当}n\geq 1\text{时}\end{cases}$$

求解阶乘函数的递归算法

```
long Factorial(long n) {
    if (n == 0) return 1;
    else return n*Factorial(n-1);
}
```

五、栈与递归

求解阶乘 n! 的过程

$$n!=\begin{cases}1, & \text{当} n=0 \text{时} \\ n*(n-1)!, & \text{当} n\geq 1 \text{时}\end{cases}$$

求解阶乘函数的递归算法

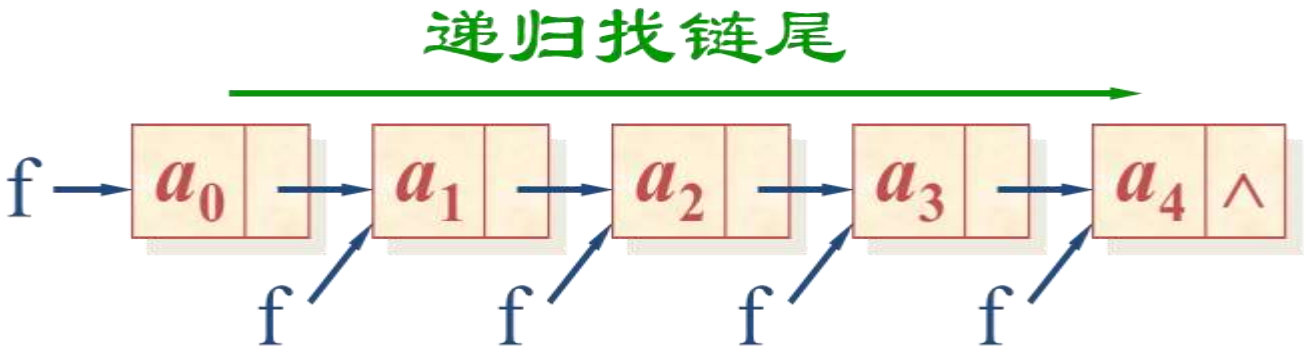
```
long Factorial(long n) {
    if (n == 0) return 1;
    else return n*Factorial(n-1);
}
```

五、栈与递归

例2：单链表结构

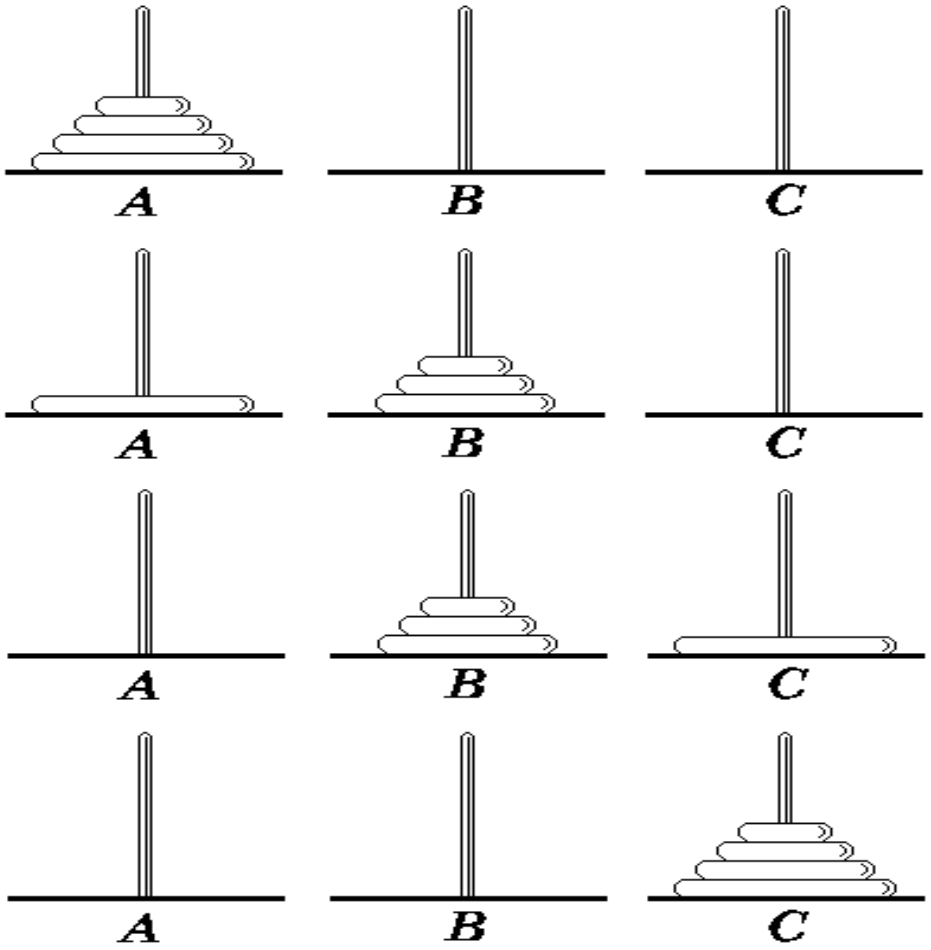
搜索链表最后一个结点并打印其数值

```
template <class E>
void Print(ListNode<E> *f) {
    if (f ->link == NULL)
        cout << f ->data << endl;
    else Print(f ->link);
}
```



五、栈与递归

例3：汉诺塔(Tower of Hanoi)问题的解法



五、栈与递归

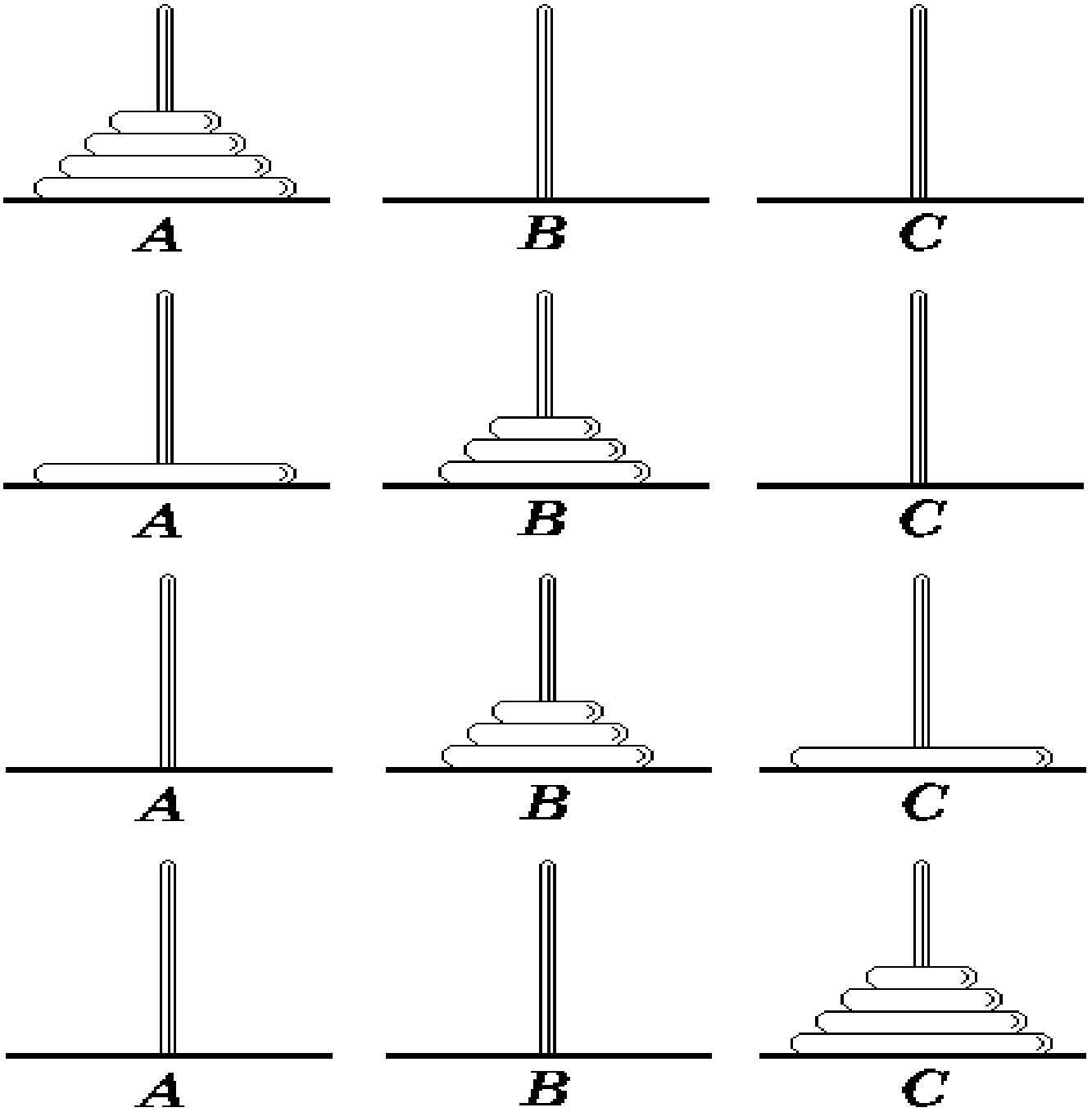
例3：汉诺塔(Tower of Hanoi)问题的解法

如果 n = 1，则将这一个盘子直接从 A 柱移到 C 柱上。否则，执行以下三步：

① 用 C 柱做过渡，将 A 柱上的 (n-1) 个盘子移到 B 柱上；

② 将 A 柱上最后一个盘子直接移到 C 柱上；

③ 用 A 柱做过渡，将 B 柱上的 (n-1) 个盘子移到 C 柱上。


A
B
C
A
B
C
A
B
C
A
B
C

```
#include <iostream.h>
void Hanoi (int n, char A, char B, char C) {
//解决汉诺塔问题的算法
    if (n == 1) cout << " move " << A << " to  "
          << C << endl;
    else {  Hanoi(n−1, A, C, B);
             cout << " move " << A << " to " << C
                   << endl;
             Hanoi(n−1, B, A, C);
          }
}
```

什么时候运用递归？

- 子问题应与原问题做同样的事情，且更为简单；
- 把一个规模比较大的问题分解为一个或若干规模比较小的问题，分别对这些比较小的问题求解，再综合它们的结果，从而得到原问题的解。

— 分而治之策略（分治法）

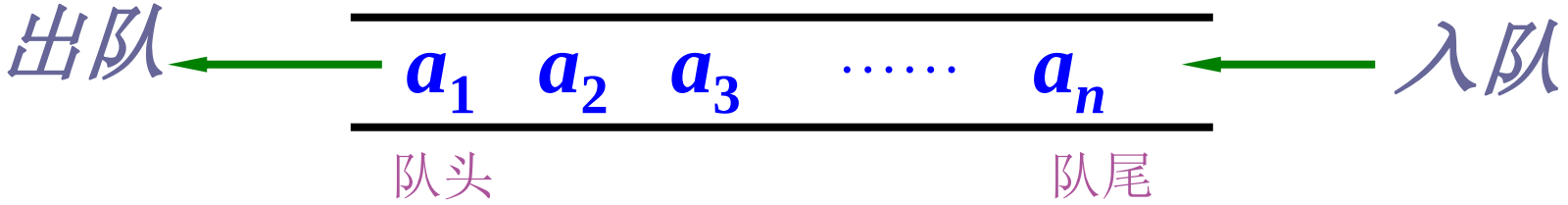
- 这些比较小的问题的求解方法与原来问题的求解方法一样。

3.4 队列

第四节　队列

一、队列

- 队列是只允许在表的一端进行插入，而在另一端删除元素的线性表。
- 在队列中，允许插入的一端叫队尾（rear），允许删除的一端称为对头(front)。
- 特点：先进先出（FIFO）



第四节　队列

二、顺序队列

- 顺序队列：采用一组地址连续的存储单元依次存储从队列头到队列尾的元素
- 顺序队列有两个指针：队头指针front和队尾指针rear

第四节　队列

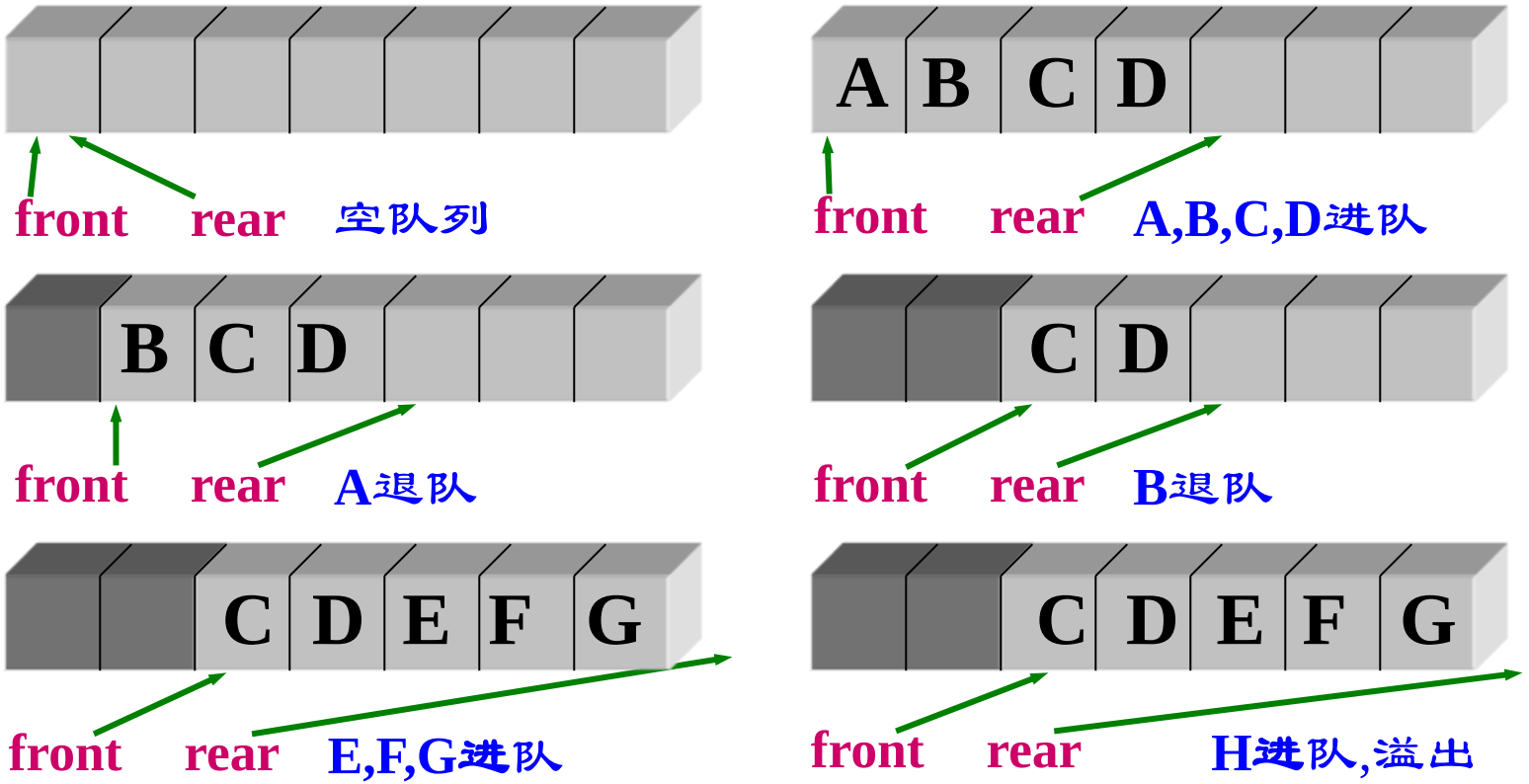
三、顺序队列的进队和出队原则

- 进队时，新元素按rear指针位置插入，然后队尾指针增一，即 rear = rear + 1
- 出队时，将队头指针位置的元素取出，然后队头指针增一，即 front = front + 1

- 队头指针始终指向队列头元素
- 队尾指针始终指向队列尾元素的下一个位置

第四节　队列

四、顺序队列的进队和出队举例



第四节　队列

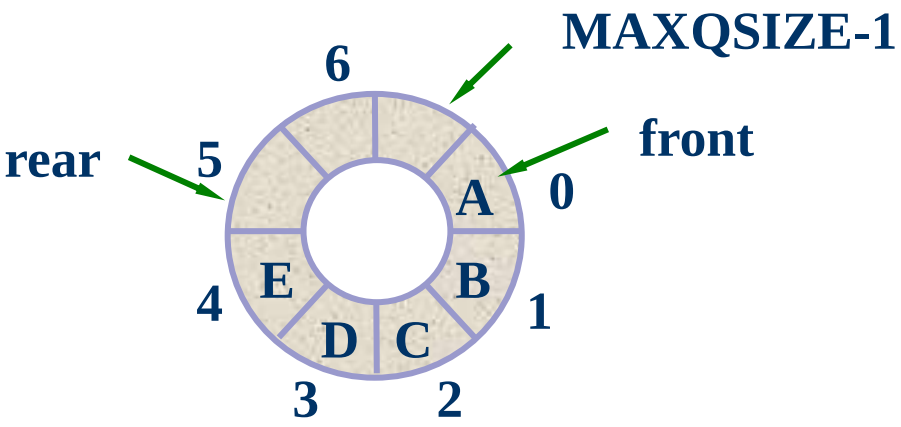
五、顺序队列存在的问题

- 当队尾指针指向队列存储结构中的最后单元时，如果再继续插入新的元素，则会产生溢出
- 当队列发生溢出时，队列存储结构中可能还存在一些空白位置（已被取走数据的元素），假上溢。
- 解决办法之一：将队列存储结构首尾相接，形成循环（环形）队列

第五节　循环队列

一、循环队列

- 循环队列采用一组地址连续的存储单元
- 将整个队列的存储单元首尾相连



这种循环意义下的加1操作可以描述为：

```
if(i+1==MAXQSIZE)
        i=0;
 else
        i++;
```

利用模运算可简化为：

```
i=(i+1)% MAXQSIZE
```

如何判断队空还是队满呢？

对循环队列而言，无法通过front==rear来判断队列“空”还是“满”。

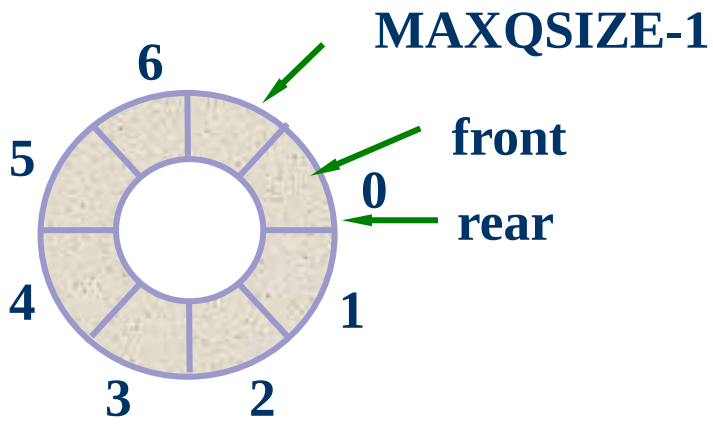
解决此问题的方法至少有三种：

- 其一是另设一个布尔变量以匹别队列的空和满；
- 其二是少用一个元素的空间，约定入队前，测试尾指针在循环意义下加1后是否等于头指针，若相等则认为队满（注意：rear所指的单元始终为空）；
- 其三是使用一个计数器记录队列中元素的总数（实际上是队列长度）。

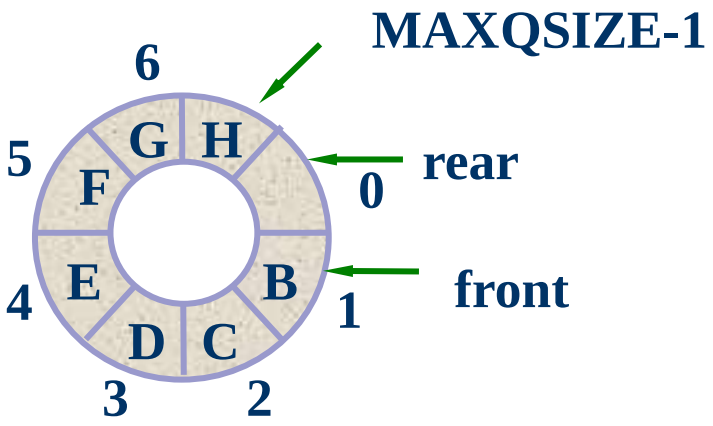
第五节　循环队列

二、循环队列空与满

- front = rear，循环队列空
- (rear+1) % MAXQSIZE = front，循环队列满



队列空

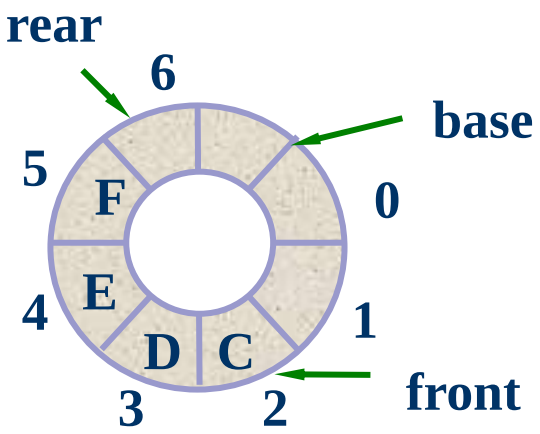


队列满

第五节　循环队列

三、循环队列定义

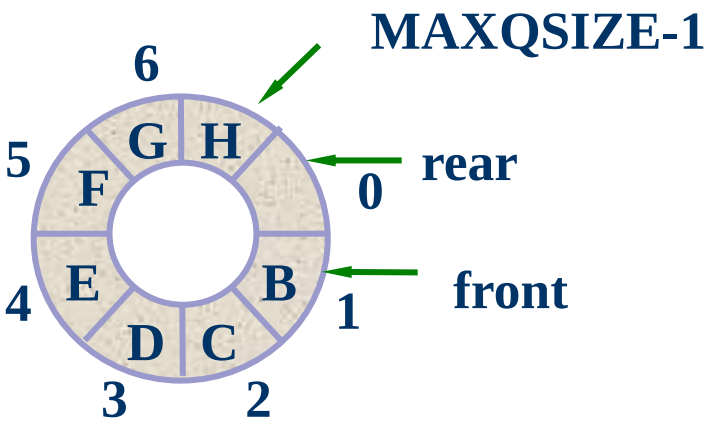
```
template <class T>
class SQueue
{
    T   *base;          //顺序存储
    int  MaxSize;       //数组大小
    int  front,rear;    //队头、队尾
public:
    SQueue();           //分配空间，置队头、队尾
    void push(T e);     //e入队列
    bool empty();       //判队空
    bool full();        //判队满
    T front();          //获得队头元素
    void pop();         //出队列
};
```



第五节　循环队列

三、循环队列插入元素

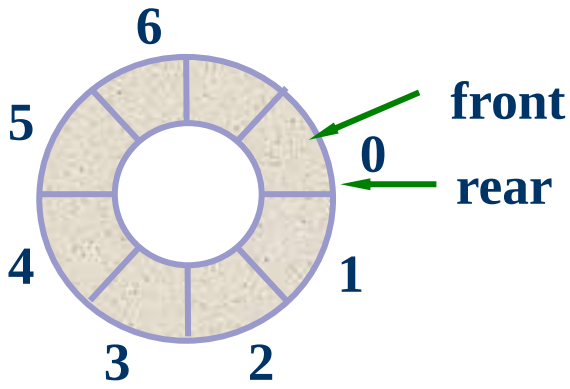
```
template <class T>
void SQueue<T>::push(T e)
{
    base[rear] = e;
    rear = (rear+1)%MaxSize;
}
```



第五节　循环队列

三、循环队列判队满

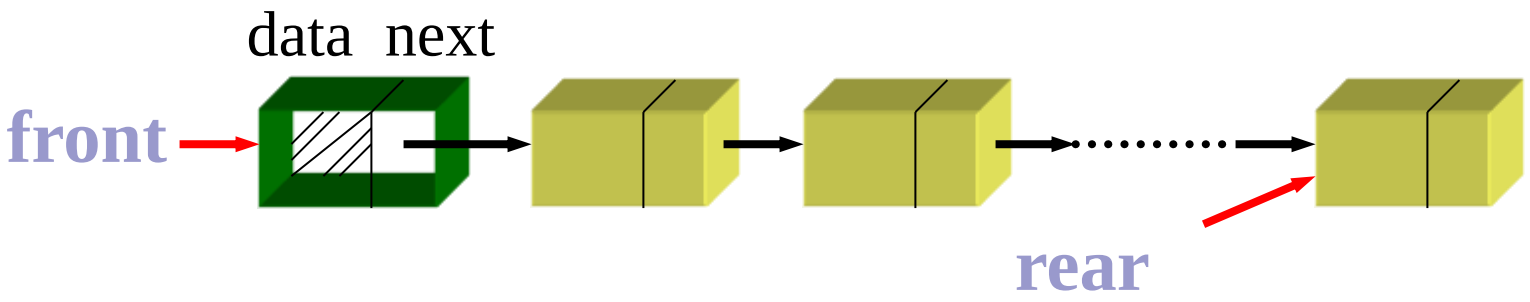
```
template <class T>
bool SQueue<T>::Full()
{
    return (rear+1)%MaxSize == front;
}
```



第六节　链队列

一、链队列

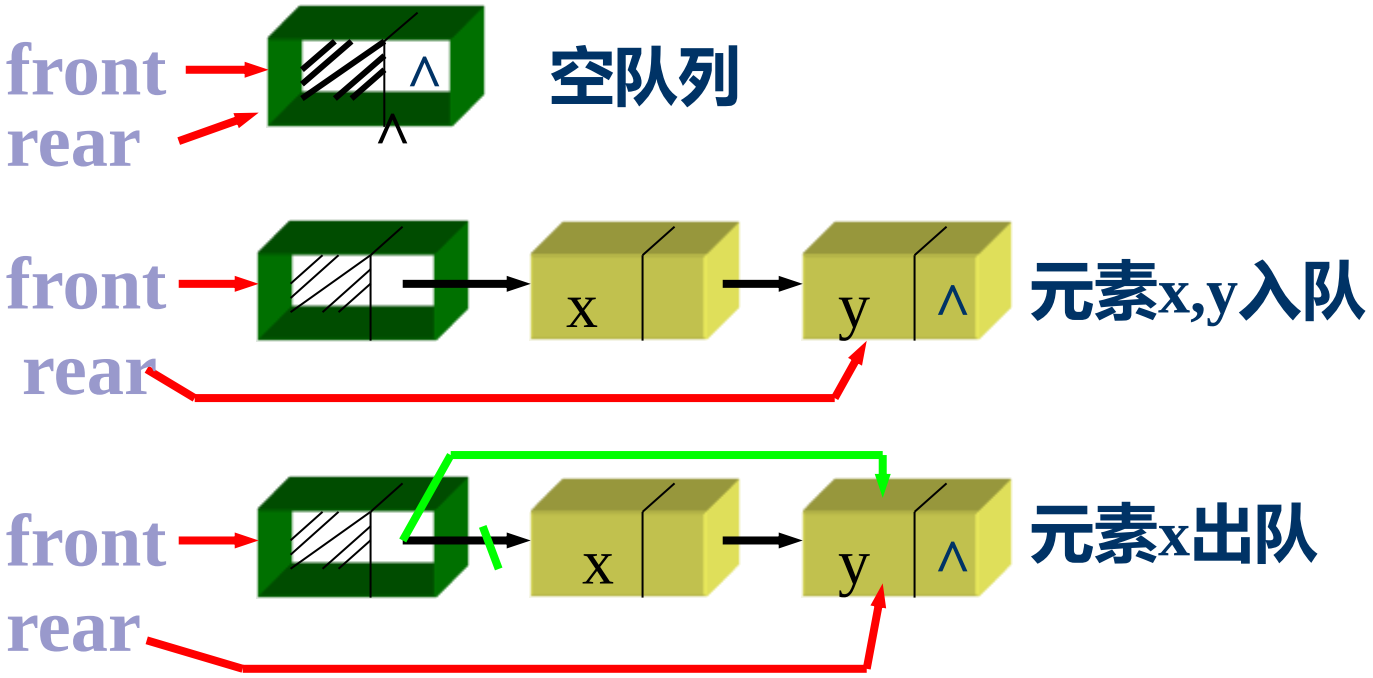
- 链队列采用链表存储单元
- 链队列中，有两个分别指示队头和队尾的指针。
- 链式队列在进队时无队满问题，但有队空问题。



第六节　链队列

二、链队列指针变化状况

- 链队列是链表操作的子集



C++中队列容器

queue 模板类的定义在**<queue>**头文件中。

定义**queue** 对象的示例代码如下：

```
queue<int> q1;
queue<double> q2;
```

queue 的基本操作有：

入队，如例：**q.push(x);** 将**x** 接到队列的末端。

出队，如例：**q.pop();** 弹出队列的第一个元素，注意，并不会返回被弹出元素的值。

访问队首元素，如例：**q.front()**，即最早被压入队列的元素。

访问队尾元素，如例：**q.back()**，即最后被压入队列的元素。

判断队列空，如例：**q.empty()**，当队列空时，返回**true**。

访问队列中的元素个数，如例：**q.size()**

C++中map容器

map是c++的一个标准容器，提供了key和value的映射。
map 模板类定义在**<map>**头文件中。

map对象定义：
map<string , int > mapstring;

map添加数据：
map<int ,string> maplive;
1.maplive.insert(pair<int,string>(102,“aclive”));
2.maplive.insert(map<int,string>::value_type(321,“hai”));
3.maplive[112]="April";//map中最简单最常用的插入添加！

作业

1、设将整数1、2、3、4依次进栈，但只要出栈时栈非空，则可将出栈操作按任何次序夹入其中，请回答下有问题：

（1）若入栈次序为push(1)，pop()，push(2），push(3)，pop()，pop( )，push(4)，pop( )，则出栈的数字序列为什么？

2、栈和队列是不是线性表？相比于线性表有什么特殊之处？

作业

3、 当利用大小为N的数组顺序存储一个栈时,假定用top==N表示栈空,则向这个栈插入一个元素时,首先应执行(　)语句修改top指针.

A. top++

B. top—

C. top=0

D. top=N-1

作业

4、假定利用数组a[N]顺序存储一个栈，top表示栈顶指针，top==-1表示栈空，并已知栈未满，当元素X进栈是所执行的操作为（）。

A. a[--top]=x

B. a[top--]=x

C. a[++top]=x

D. a[top++]=x

作业

5、判定一个栈s（最多元素数为m0）为空的条件是（ ）。为满的条件是（）。

A. s->top！=0

B. s->top==0

C. s->top！=m0

D. s->top==m0

E. s->top！=m0-1

F. s->top==m0-1

作业

6、单链表实现的栈，栈顶指针为Top(仅仅是一个指针)，入栈一个P节点时，其操作步骤为：（ ）。

A. Top->next=p

B. p->next=Top->next; Top->next=p

C. p->next=Top; Top=p->next

D. p->next=Top; Top=Top->next

作业

7、循环队列的优点是什么？如何判断它的空和满？

8、设长度为n的链队列用单循环链表表示，若只设头指针，则怎样进行入队和出队操作；若只设尾指针呢？

作业

9、一个队列的入队序列是1，2，3，4，则队列的输出序列是（）

A. 4，3，2，1

B. 1，2，3，4

C. 1，4，3，2

D. 3，2，4，1

作业

10、假定一个顺序循环队列的队首和队尾指针分别用front和rear表示，则判断队空的条件为（）。

A. front +1==rear

B. rear+1==front

C. front= =0

D. front= =rear

作业

11、假定一个顺序循环队列存储于数组a[N]中，其队首和队尾指针分别用front和rear表示，则判断队满的条件为（）。

A. （rear-1）%N= =front

B. （rear+1）%N==front

C. （front-1）%N== rear

D. （front+1）%N==rear

作业

12、循环队列用数组a[m]存放其元素值，已知其头尾指针分别用front和rear表示，则队列中的元素个数为（）。

A. （rear-front+m）%m

B. rear-front+1

C. rear-front-1

D. rear-front

作业

13、假定一个链队列的队首和队尾指针分别用front和rear表示，每个结点包含data和next两个域，出队时所进行的指针操作为（）。

A. front=front->date

B. front=front->next

C. rear=rear->next

D. rear=rear->date

作业

14、设栈S和队列Q的初始状态为空，元素e1，e2，e3，e4，e5，e6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e2，e4，e3，e6，e5，e1，则栈S的容量至少应该为_____。

A. 2　　B. 3

C. 4　　D. 5