

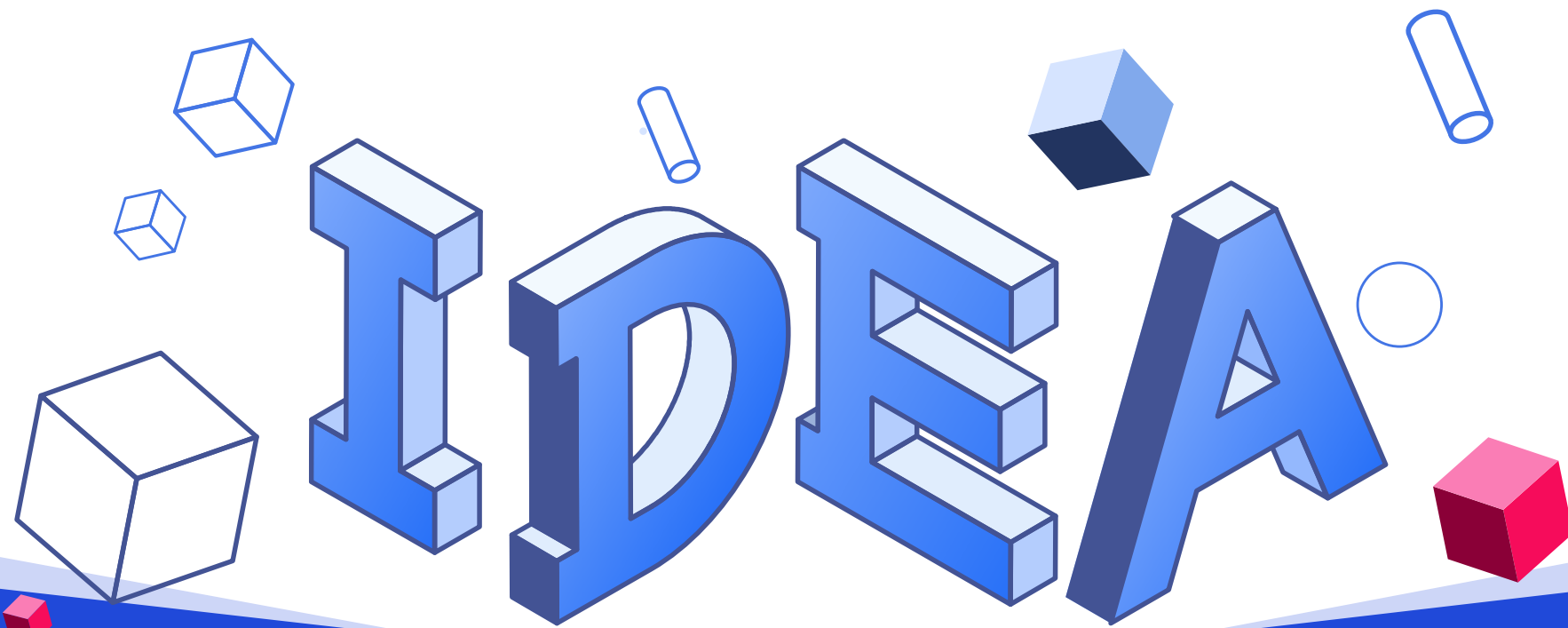
编程学什么？ {
 程序语言
 算法
 数据结构

数据结构

简单来说就是组织数据的方式

通过精心、合理、结构化的组织数据，可以带来更高的程序运行或者数据存储效率

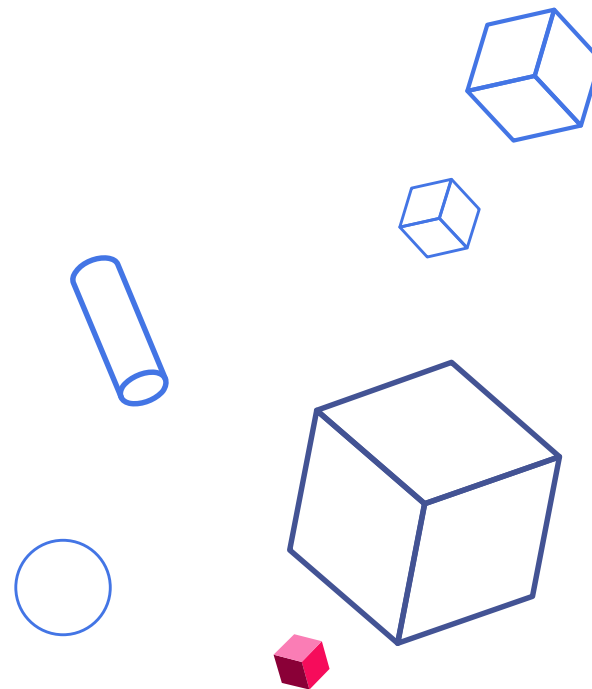
例：数组就是大家接触到的第一个基础数据结构，通过顺序的存储、下标的访问这样一些方式组织了我们的数据。这样的数据结构，可以使我们 $O(1)$ 访问数据， $O(n)/O(\log n)$ 查询数据...



信息学 队列、栈

西南大学附属中学校
信息奥赛教练组

01 队列(queue)





队列

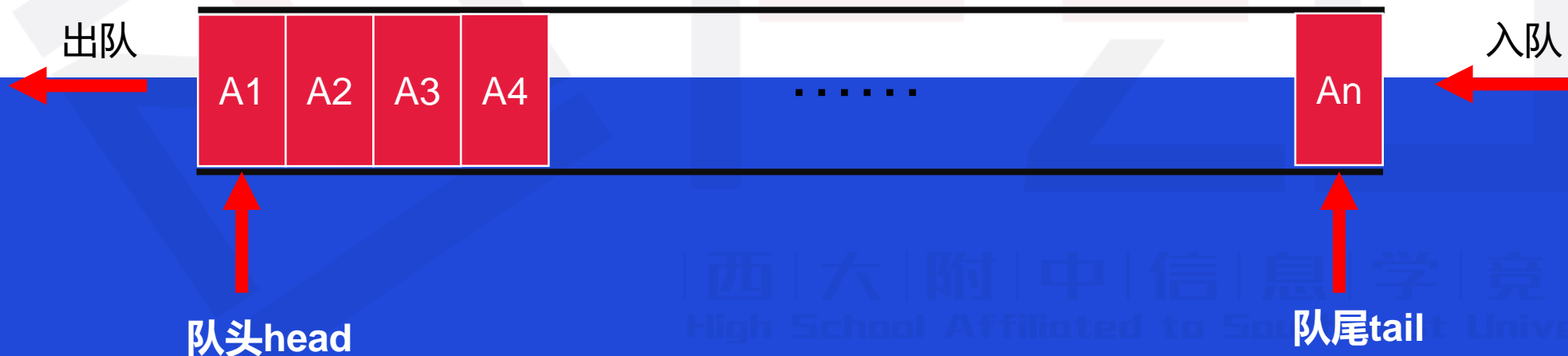


西南大学附属中学
High School Affiliated to Southwest University

跟我们生活中排队取票，排队结账是一样的。

定义： 队列是限定在一端进行插入，另一端进行删除的一种特殊线性表

在**队头**进行**删除**，称为出队；
在**队尾**进行**插入**，称为入队。



队列的特征是：**先进先出**

队列也是一种线性的数据结构，所以也可以用数组模拟实现

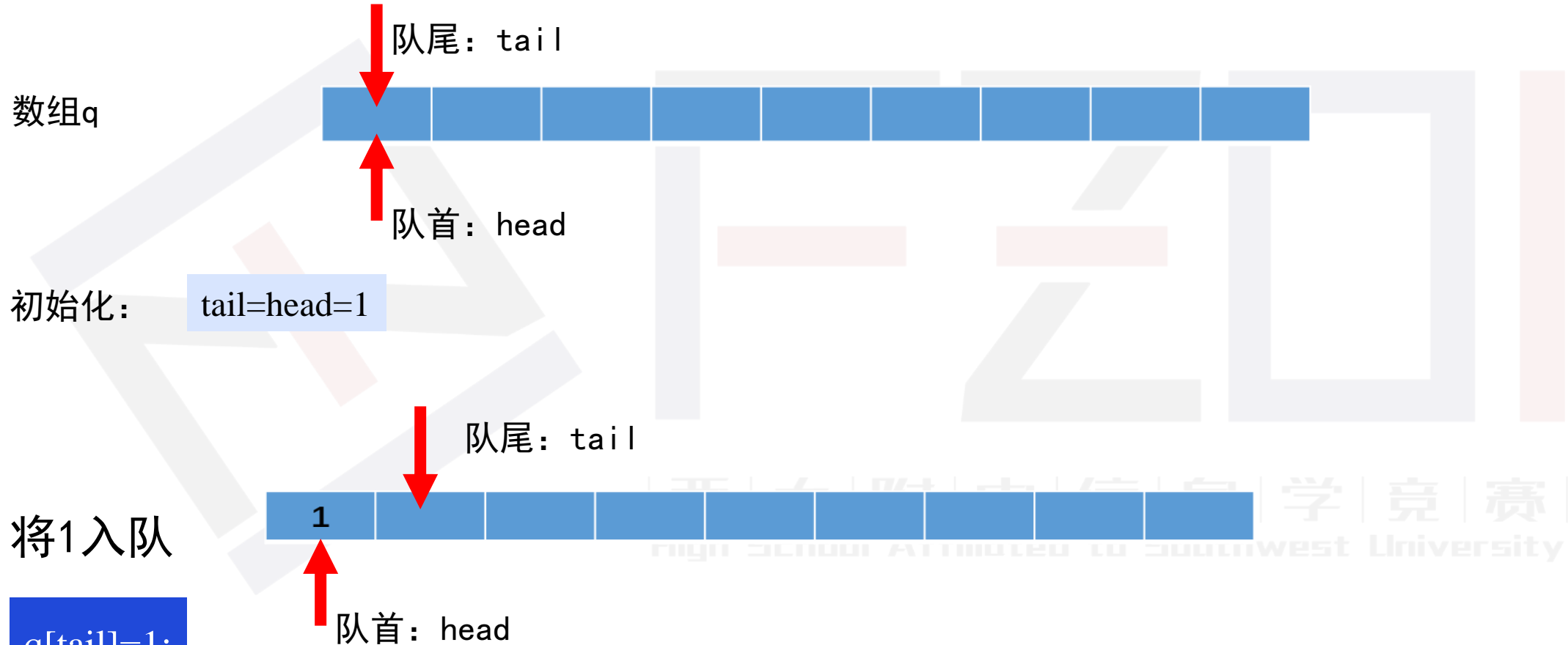


入队操作



西南大学附属中学
High School Affiliated to Southwest University

例：让1 2 4 5四个元素入队



```
q[tail]=1;  
tail++;
```



入队操作



西南大学附属中学
High School Affiliated to Southwest University

将4入队

```
q[tail]=4;  
tail++;
```



将2入队

```
q[tail]=2;  
tail++;
```



西 | 大 | 附 | 中 | 信 | 息 | 学 | 竞 | 赛 |
High School Affiliated to Southwest University



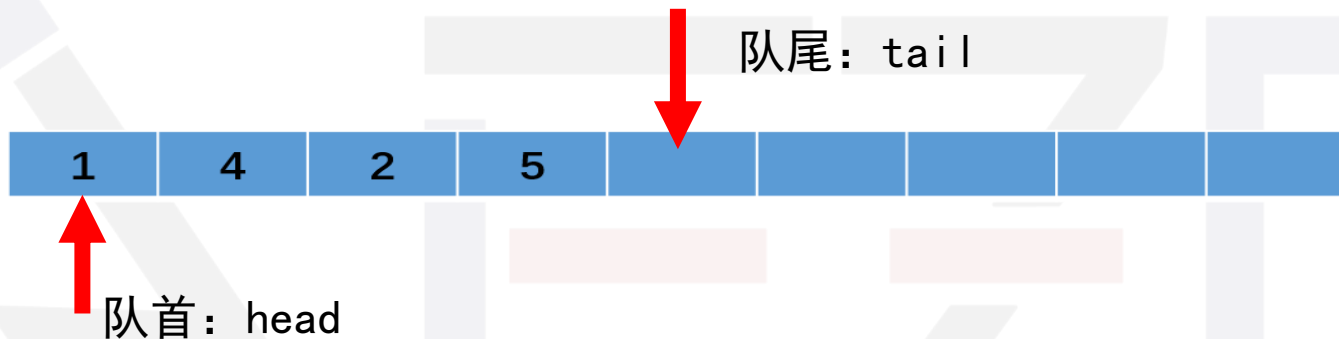
入队操作



西南大学附属中学
High School Affiliated to Southwest University

将5入队

```
q[tail]=5;  
tail++;
```



入队操作: $q[tail++] = x;$

出队操作: 移动队头, $head++;$



队列的其他操作



西南大学附属中学
High School Affiliated to Southwest University

初始化

`head=tail=1;`

入队

`q[tail++]=x;`

出队

`head++;`

访问队头元素

`q[head];`

访问队尾元素

`q[tail-1];`

队列中元素个数

`tail-head;`

队空

`tail == head;`

队满

`tail==maxn;`

STL也提供了queue容器，同样定义了队列相关操作的函数

`#include <queue>`

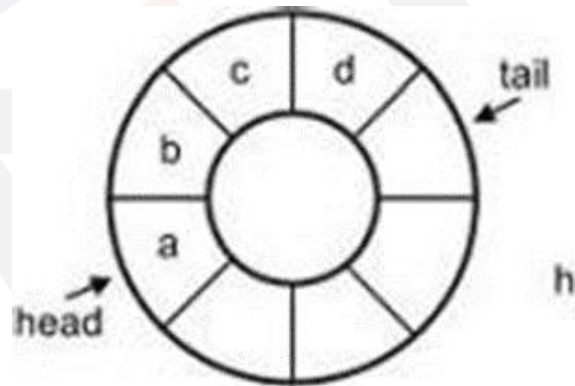
`queue<数据类型> q;`

<code>x=q.front()</code>	: 得到队头元素
<code>x=q.back()</code>	: 得到队尾元素
<code>q.push(x)</code>	: x入队
<code>q.pop()</code>	: 队头出队，无返回值
<code>q.empty()</code>	: 队空返回1，否则返回0
<code>q.size()</code>	: 得到队列的大小(长度)

西 | 大 | 附 | 中 | 信 | 息 | 学 | 竞 | 赛 |
High School Affiliated to Southwest University

可以发现，当随着不断出队，前面会空出许多空间；而当有新的入队时，那些空间是无法被利用的。如何提高空间利用率呢？

解决办法：把队列做成一个环，成为循环队列



(a)一般情况

循环队列入队：

$$q[\text{tail}] = x;$$
$$\text{tail} = (\text{tail} + 1) \% \text{maxn};$$

循环队列出队：

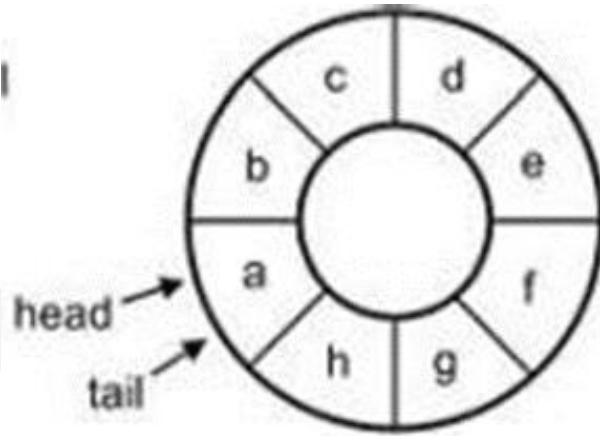
$$\text{head} = (\text{head} + 1) \% \text{maxn}$$



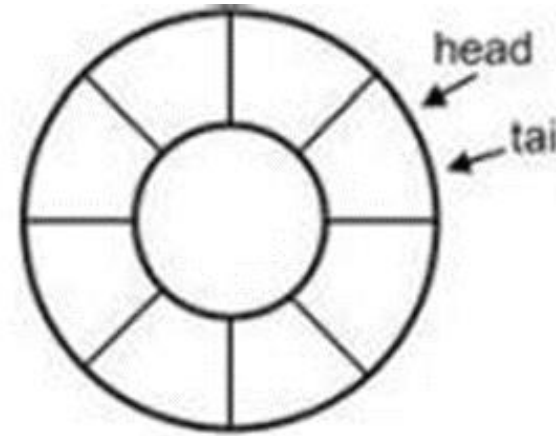
循环队列



西南大学附属中学
High School Affiliated to Southwest University



(b)队满



(c)队空

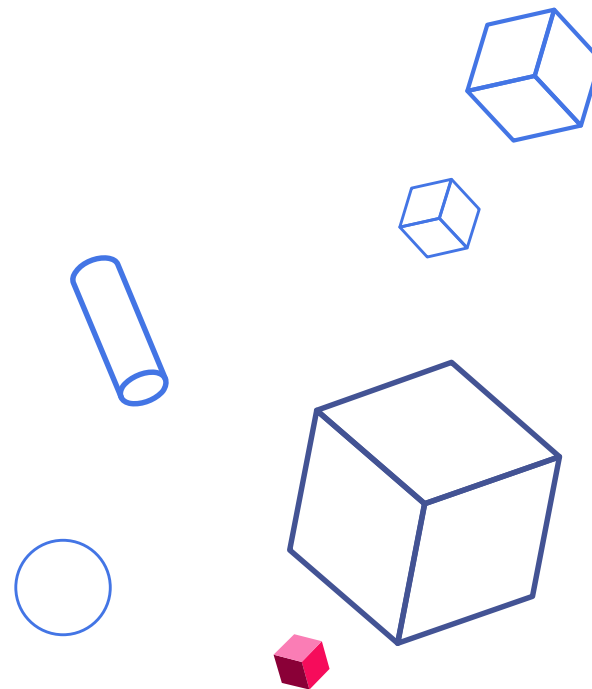
循环队列队满:

$$(tail + 1) \% maxn == head \% maxn$$

循环队列队空:

$$head == tail$$

02 栈(stack)



定义：是只在表的一端进行插入和删除运算的线性表

生活中栈的例子： 将一些收纳盒整理到一个大箱子里

Q:如果我们现在需要的东西在收纳盒A里面，我们该怎么办？



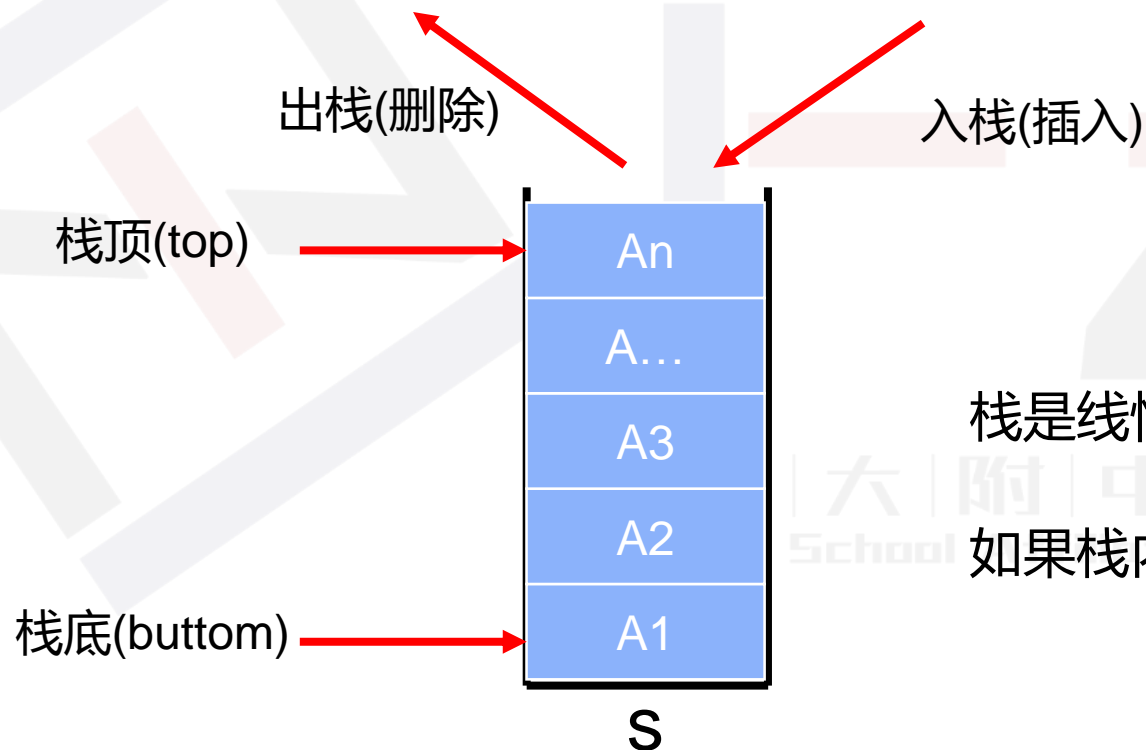
先把C拿出来，再把B拿出来，最后拿到A(C B A)

而我们放入的时候是这样的：先放入A，再放入B，最后放入C(A B C)

栈的特性：先进后出



栈只允许在栈顶一端进行插入和删除操作，我们称之为入栈和出栈。



栈是线性的数据结构，可以用数组模拟实现

如果栈内元素是递增或递减的，称为**单调栈**



栈的操作



西南大学附属中学
High School Affiliated to Southwest University

初始化

`top=0;`

入栈

`s[++top]=x;`

出栈

`top--;`

访问栈顶元素

`s[top];`

判断栈满

`if(top==maxn)`

判断栈空

`if(top==0)`

栈内元素个数

`top`

STL提供了stack栈容器，里面定义了这些基本操作的函数。

`#include <stack>`

`stack<数据类型> s;`

`x=s.top()` : 得到栈顶元素

`s.push(x)` : x入栈

`s.pop()` : 栈顶出栈，无返回值

`s.size()` : 栈大小

`s.empty()` : 栈空为1，否则为0

栈的应用：ctrl+Z、递归、检验括号匹配、求后缀表达式、中缀转后缀等



括号匹配



西南大学附属中学
High School Affiliated to Southwest University

假设表达式中允许圆括号 () 和方括号 [] 两种括号，其嵌套的顺序随意，如 ([]) 或 [()] [([])] 等为正确匹配，[(]) 或 ([()]) 等为错误匹配。现在给你一个表达式，检验括号是否正确匹配。

输入

一行字符，包含圆括号和方括号，长度小于255。

输出

匹配输出 "OK", 不匹配输出 "Wrong".

样例输入

[(]

样例输出

Wrong

西 | 大 | 附 | 中 | 信 | 息 | 学 | 竞 | 赛
High School Affiliated to Southwest University

每种括号都有左右括号，如果匹配，左右括号的个数应该是一致的，可以想成出栈和入栈的次数一致的情况。

用栈模拟：

1.遇到左括号

将左括号入栈

2.遇到右括号

(1) 栈顶左括号出栈

但要首先判断是否与栈顶元素匹配

如果匹配，将栈顶左括号出栈，如果不匹配，则字符串不匹配，结束。

(2) 如果栈为空，也不匹配

3.处理完后，检查栈内元素情况

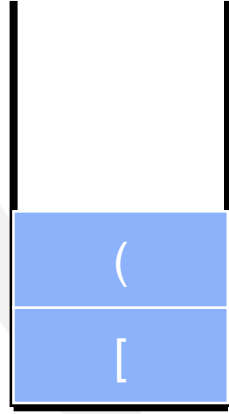
(1) 如果栈非空，则左括号多了，不匹配

(2) 如果栈为空，个数一致，匹配

[()]



左括号[



左括号(



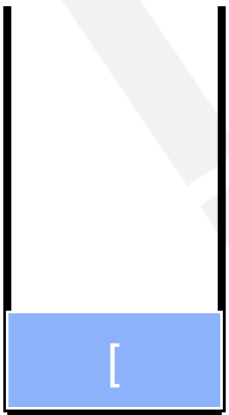
右括号), 匹配, 出栈



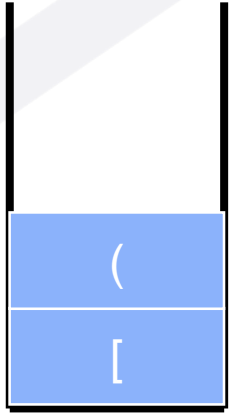
右括号], 匹配, 出栈

栈空且无
失配, 匹
配成功

[(])



左括号[



左括号(



右括号], 失配, 结束



核心代码



西南大学附属中学
High School Affiliated to Southwest University

```
char a[260],s[260];
scanf("%s",a);
len=strlen(a);
i=0;top=0;flag=1;
while(i<len&&flag) //如果没处理完并且匹配过程中没有失配的情况
{
    if(a[i]=='['||a[i]=='(') //遇到左括号入栈
        s[++top]=a[i];
    if(a[i]==']') //遇到右括号出栈
        if(s[top]=='[') top--; //判断是否为相应的右括号
        else flag=0;
    if(a[i]==')')
        if(s[top]=='(') top--;
        else flag=0;
    i++;
}
if(flag&&top==0) printf("OK\n"); //处理完毕过程中没有失配，并且栈空
else printf("Wrong\n");
```



后缀表达式



西南大学附属中学
High School Affiliated to Southwest University

从键盘读入一个后缀表达式（字符串），只含有0-9组成的运算数及加（+）、减（—）、乘（*）、除（/）四种运算符。每个运算数之间用一个空格隔开，不需要判断给你的表达式是否合法。以@作为结束标志。

后缀表达式概念：

不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右进行（不再考虑运算符的优先规则，

如： **$(2 + 1) * 3$** ，**后缀表达式为** $2\ 1\ +\ 3\ *$

输入

一行字符串（不超过255个字符），值包含1到9，+、-、*、/，并

以@结束

输出

表达式的值

样例输入

16 9 4 3 + * - @

样例输出

-47

西|大|附|中|信|息|学|竞|赛
High School Affiliated to Southwest University




分析



西南大学附属中学
High School Affiliated to Southwest University

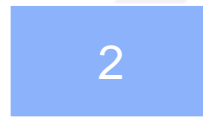
2 1 + 3 * 的计算过程: $2+1=3$



$3*3=9$

思路: 遍历字符串,

- 如果是数字, 将数字入栈;
- 如果是操作符, 就出栈两个数字, 进行运算, 运算的结果再入栈。





拓展：中缀表达式计算



西南大学附属中学
High School Affiliated to Southwest University

后缀表达式是计算机比较喜欢的计算方式，但是在日常生活中人们常用中缀表达式。

但中缀表达式不再符合我们刚才的计算方式，所以一般会先将中缀表达式转换为一个后缀表达式。

中缀转后缀的方法：

- 1.遇到操作数，直接输出(或保存)；
- 2.栈为空时，遇到运算符，入栈；
- 3.遇到左括号，将其入栈；
- 4.遇到右括号，执行出栈操作，并将出栈的元素输出，直到弹出栈的是左括号，左括号不输出；
- 5.遇到其他运算符' + "- "*" "/" 时，弹出所有**优先级**大于或等于该运算符的栈顶元素，然后将该运算符入栈；
- 6.最终将栈中的元素依次出栈，输出。

关键点在于确定各个符号之间的优先级关系(打表)



执行过程

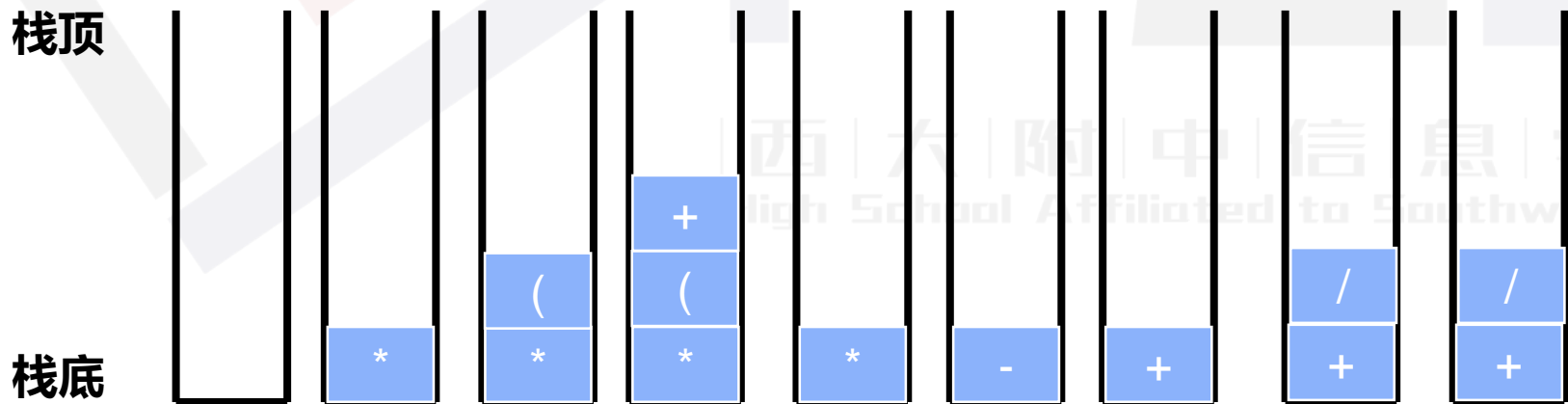


西南大学附属中学
High School Affiliated to Southwest University

1. 遇到操作数，直接输出(或保存);
2. 栈为空时，遇到运算符，入栈;
3. 遇到左括号，将其入栈;
4. 遇到右括号，执行出栈操作，并将出栈的元素输出，直到弹出栈的是左括号，左括号不输出;
5. 遇到其他运算符' + "- "*" "/" 时，弹出所有**优先级**大于或等于该运算符的栈顶元素，然后将该运算符入栈;
6. 最终将栈中的元素依次出栈，输出。

中缀 12 * (3 + 4) - 6 + 8 / 2

后缀 12 3 4 + * 6 - 8 2 / +



图文解释: https://blog.csdn.net/qq_34992845/article/details/70313588

Thanks

For Your Watching

