

# 数据结构

编程学什么？

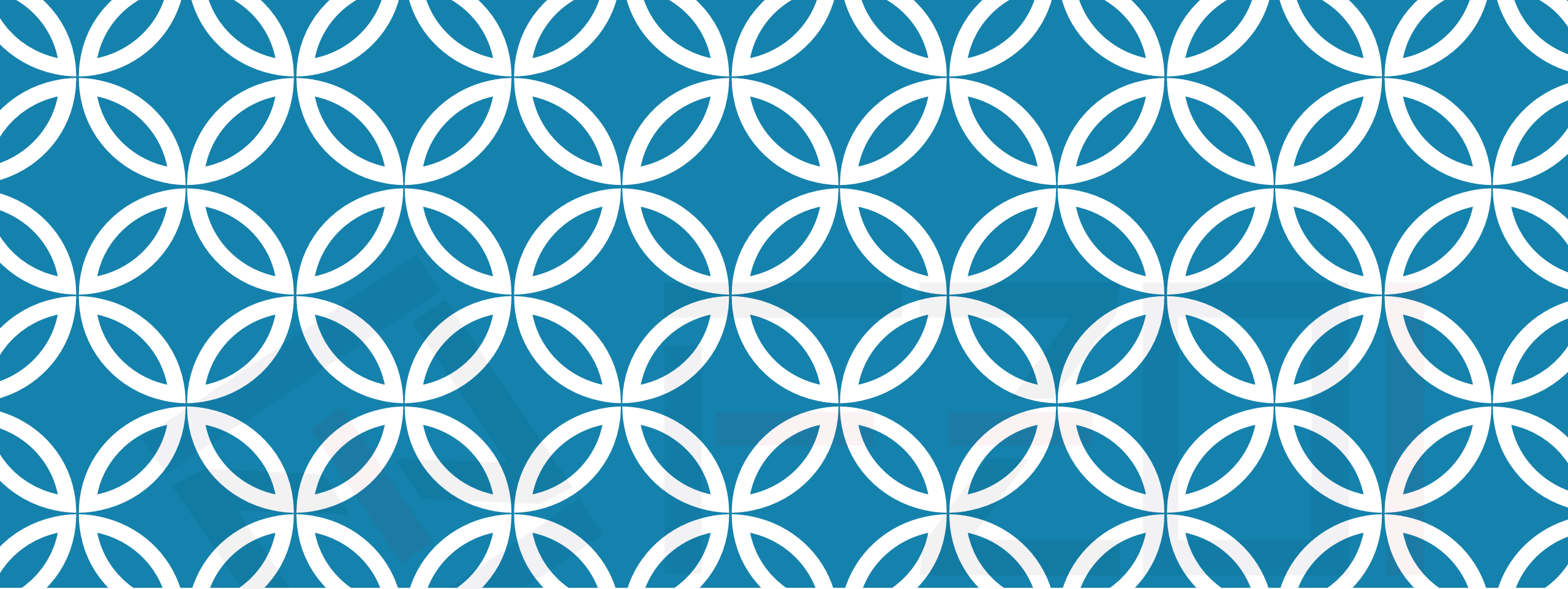
- 程序语言
- 算法
- 数据结构

## 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...



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

# 数据结构之链表

QYC

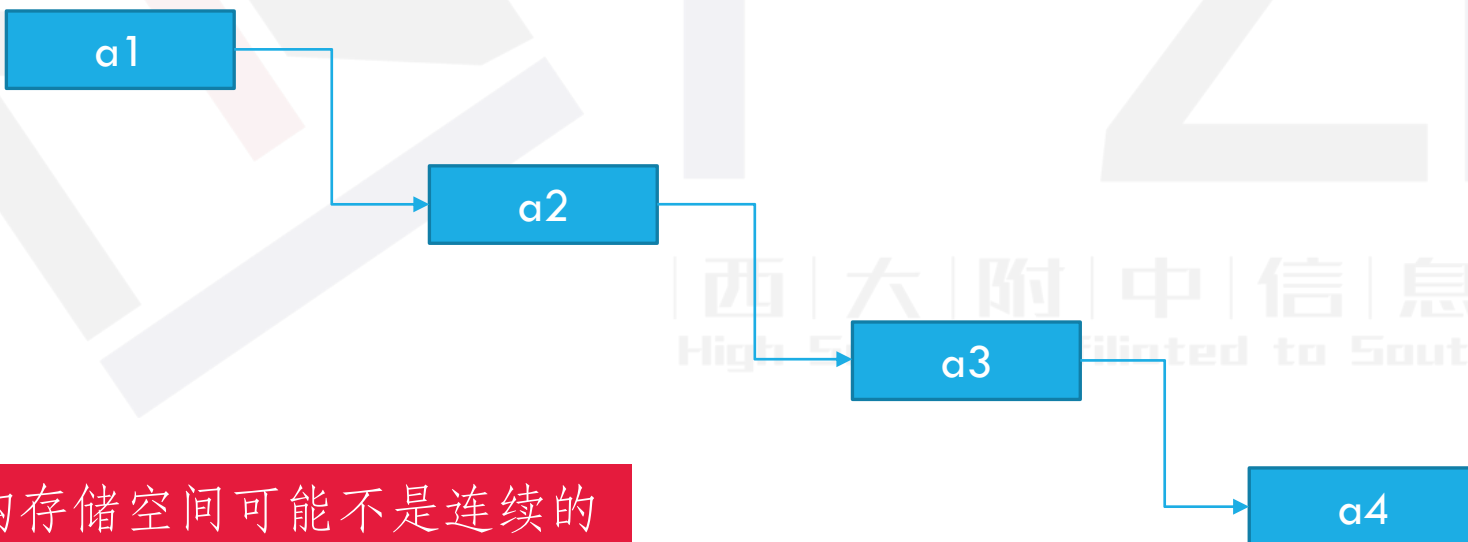
# 什么是链表?

存储结构分为顺序存储结构和链式存储结构，顺序存储结构如数组，链式存储结构如链表。

数组:



链表:



链式结构存储空间可能不是连续的

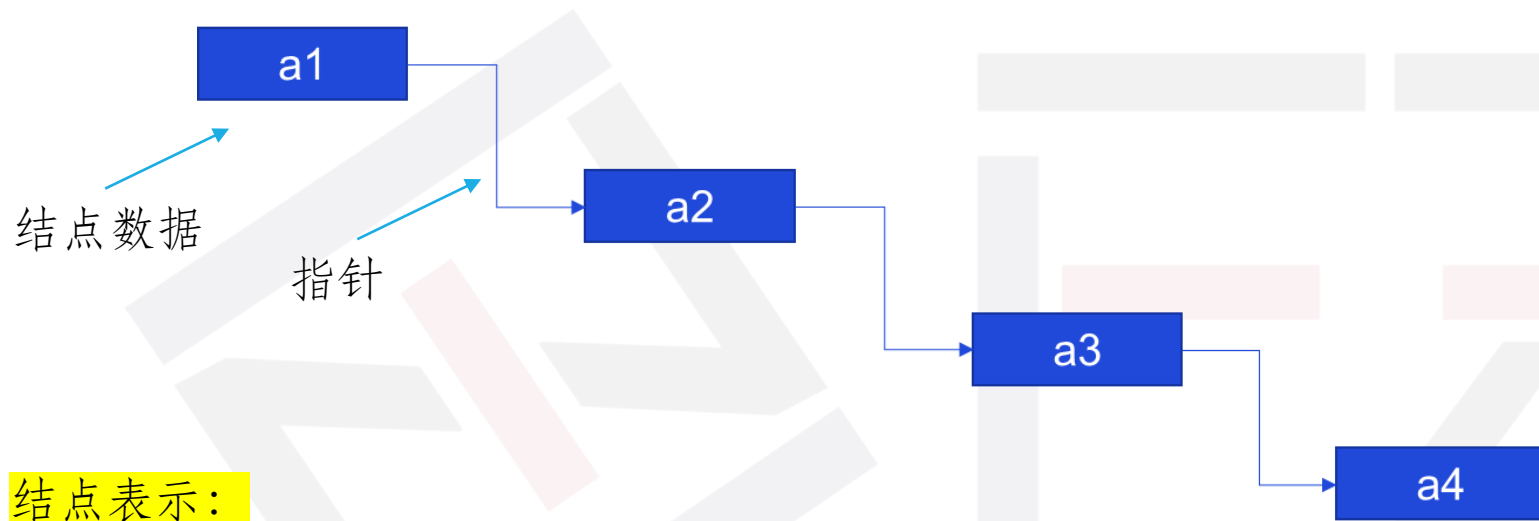
# 静态和动态链表的区别

- 1、**静态链表**是用数组方法实现的，是顺序的存储结构，在物理地址上是连续的，而且需要预先分配地址空间大小。所以静态链表的初始长度一般是固定的，在做插入和删除操作时不需要移动元素，仅需修改指针。
- 2、**动态链表**是用内存申请函数（malloc/new）动态申请内存的，所以在链表的长度上没有限制。动态链表因为是动态申请内存的，所以每个节点的物理地址不连续，要通过C指针(自行了解)来顺序访问。

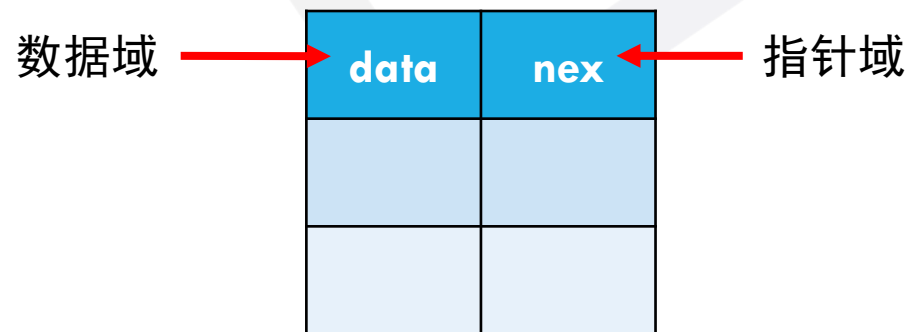
# 链表与数组

	数组	动态链表	静态链表
内存存储	连续，下标访问	不连续	连续
大小	固定	可变	固定
插入时间复杂度	$O(N)$	$O(1)$	$O(1)$
删除时间复杂度	$O(N)$	$O(1)$	$O(1)$
修改时间复杂度	$O(1)$	$O(N)$	$O(N)$
查找时间复杂度	$O(1)$	$O(N)$	$O(N)$

# 静态链表的表示



结点表示:



对于每个元素之间，处理存储本身的信息（**数据域**）外，还要存储它的后继元素、前驱元素的存储位置（**指针域**），这些信息合在一起称为一个结点。

1. 数组表示:

数组data[], nex[]

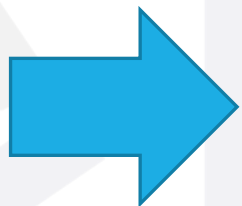
2. 结构体表示:

```
struct{  
    int data;  
    int nex;  
}a[];
```

# 静态链表

3	7	9	8	4
---	---	---	---	---

下标	data	nex
1	3	2
2	7	3
3	9	4
4	8	5
5	4	-1



下标	data	nex
0	head	1
1	3	2
2	7	3
3	9	4
4	8	5
5	4	-1

链表的起始点：头结点  
每一个链表都需要一个头结点

指针域nex指向下一个数的下标

# 链表存储

下标

0

1

2

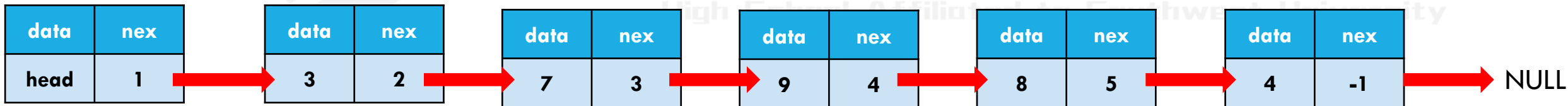
3

4

5

data	nex
head	1
3	2
7	3
9	4
8	5
4	-1

头结点





# 如何建立链表

数组data[], nex[]

下标	data	nex
0	head	-1
1		
2		
3		
4		
5		

head=0;  
nex[head]=-1;  
cnt=0;//当前结点下标

添加数1:

下标	data	nex
0	head	1
1	3	-1
2		
3		
4		
5		

data[1]=3;  
nex[1]=nex[cnt];  
nex[cnt]=1;  
cnt=1;

# 如何建立链表

数组data[], nex[]

下标	data	nex
0	head	1
1	3	2
2	7	-1
3		
4		
5		

```
data[2]=7;  
nex[2]=nex[cnt];  
nex[cnt]=2;  
cnt=2;
```

## 过程:

- 1.建立数据域data[]和指针域nex[]
- 2.初始化头结点信息

```
head=0;  
nex[head]=-1;  
cnt=0;//当前结点下标
```

- 3.将新增的元素链接到链表末尾

# 建立链表

第i个点

data	nex
4	-1

第cnt个点

data	nex
8	-1

data	nex
8	5

data	nex
4	-1



```
for(i=1;i<=n;i++){  
    nex[i]=nex[cnt];  
    nex[cnt]=i;  
    cnt=i;  
}
```

下标

0

1

2

3

4

5

	data	nex
0	head	1
1	3	2
2	7	3
3	9	4
4	8	5
5	4	-1

# 遍历链表

链表的第一个结点为头结点

链表的最后一个结点指针域为空（NULL），可以设为-1

```
for(i= nex[head] ; i!=-1 ; i=nex[i] )  
{  
    判断：if(data[i]满足条件) .....  
    输出....  
}
```

下标

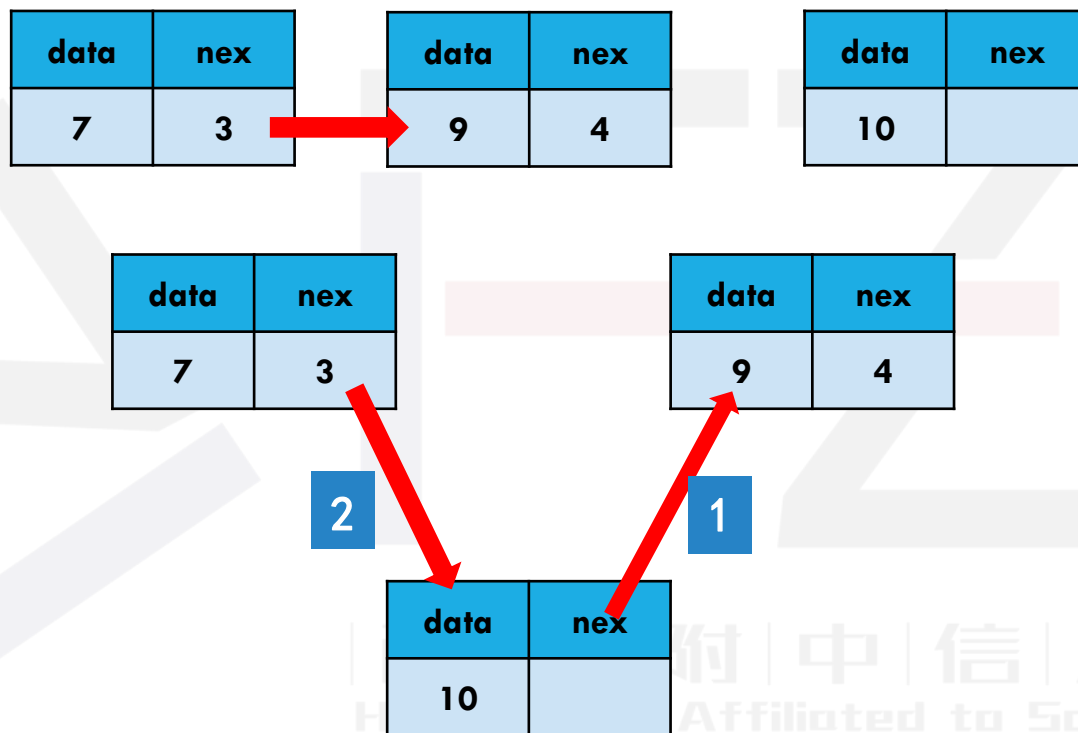
	data	nex
0	head	1
1	3	2
2	7	3
3	9	4
4	8	5
5	4	-1

# 链表—插入操作

例：在7后插入10

下标

	data	nex
0	head	1
1	3	2
2	7	3
3	9	4
4	8	5
5	4	-1



此时序列为：

3 7 10 9 8 4

下标

	data	nex
0	head	1
1	3	2
2	7	6
3	9	4
4	8	5
5	4	-1
6	10	3

# 插入过程

下标

	data	nex
0	head	1
1	3	2
2	7	6
3	9	4
4	8	5
5	4	-1
6	10	4

在第k个结点后插入s

数s添加到数组后面data[n+1]、nex[n+1]

找到第k个结点下标为p

n++; data[n]=s;

nex[n]=nex[p];


nex[p]=n;

# 链表—删除操作


例：删除9

下标	data	nex
0	head	1
1	3	2
2	7	3
3	9	4
4	8	5
5	4	-1

data	nex
7	3




data	nex
9	4



data	nex
8	5

data	nex
7	3



data	nex
9	4

data	nex
8	5

`nex[2]=nex[nex[2]];`

此时序列为：

3 7 8 4

下标	data	nex
0	head	1
1	3	2
2	7	4
3	9	4
4	8	5
5	4	-1

# 删除过程

下标

0

1

2

3

4

5

data	nex
head	1
3	2
7	3
9	4
8	5
4	-1

删除第k个结点

找到第k-1个结点下标为p，p的指针域指向k

$nex[p] = nex[nex[p]]$



# 例题：链表练习

有一个序列，包含 $n$ 个数，将 $n$ 个数使用链表存储，现有插入和删除两种操作。

输入

第一行 $n, m$  ( $n, m \leq 10000$ )

第二行 $n$ 个数

接下来 $m$ 行，每行两个数 $a, b$ 或者三个数 $a, b, c$ 。 $a=1$ 表示删除序列中的第 $b$ 个数； $a=2$ 表示在序列中的第 $b$ 个结点后插入数 $c$ 。

输出

输出 $m$ 次操作后的序列。

样例

样例输入1

5 2

2 8 7 9 1

1 3

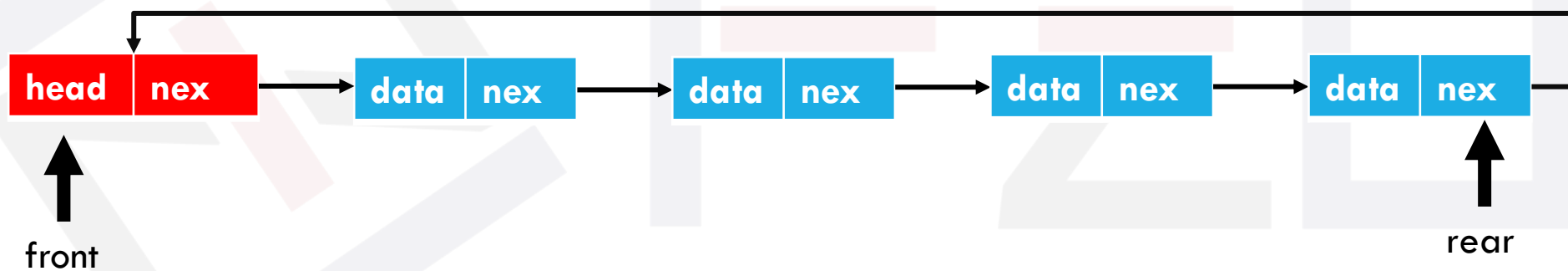
2 3 12

样例输出1

2 8 9 12 1

# 循环链表

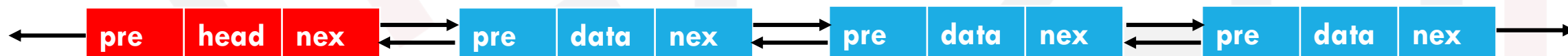
表中最后一个结点的指针域指向头结点，整个链表形成一个环，称为循环链表



# 双向链表

每一个结点只包含一个指针域，称为单链表。

每个结点有两个指针域和若干个数据域，其中一个指针指向它的前驱结点，一个指向它的后继结点，称为双链表。



# 双向链表

下标	data	nex
0	head	1
1	3	2
2	7	6
3	9	4
4	8	5
5	4	-1
6	10	4

多了一维pre

下标	pre
0	-1
1	0
2	1
3	2
4	6
5	4
6	2

# 块状链表

块状链表是结合了数组和链表的优缺点，块状链表本身是一个链表，但是链表储存的并不是一般的数据，而是由这些数据组成的顺序表。

分块的思想，均摊时间复杂度 $O(\sqrt{n})$



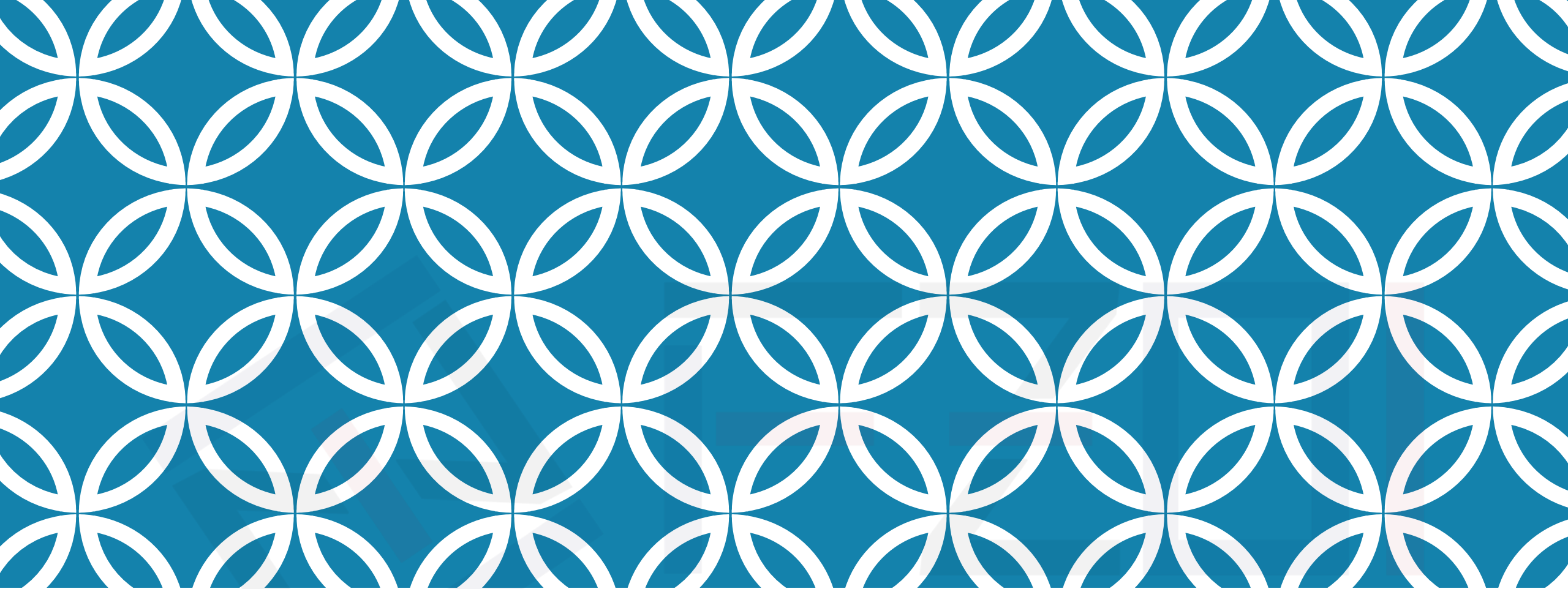
具备的操作：插入、删除、合并、查询  
进行过程中需要不断合并保证块的大小，即调整块的大小始终维持在最优的大小，保证操作的时间复杂度。

```
#include<ext/rope>
```

```
using namespace __gnu_cxx;
```

块状链表STL :rope< 数据类型> 名字

参考博客：<https://www.cnblogs.com/sduwh/p/14008253.html>



THANKS

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