第五章 位运算与集合

为什么要学习位运算?

理　　由

1、在计算机程序中，数据的位（bit）是可以操作的最小数据单位，理论上，可以用“位运算”来完成所有的运算和操作，比如：加、减、乘、除、取模等运算。

2、一般的位操作是用来控制计算机硬件的，或者进行数据变换，但是，灵活的位操作可以有效地提高程序运行的效率，比如：使用位操作可以减少除法和取模的运算。

3、在C++语言中，可以单独操控变量的位，一般高级语言不会处理这级别的细节，C++在提供高级语言便利的同时，还能在汇编语言所保留的级别上工作，这使其成为编写设备驱动程序和嵌入式代码的首选语言。

4、位运算处理速度非常快。任何信息在计算机中都是采用二进制表示的,数据在计算机中是以补码形式存储的,位运算就是直接对整数在内存中的二进制位进行运算。由于位运算直接对内存数据进行操作,不需要转换成十进制,因此处理速度非常快。

5.1 位运算

计算机储存器用二进制表示，每 8 位 bit 为 1 个字节 bite

位运算符

&	按位与	&=	按位与赋值
|	按位或	|=	按位或赋值
^	按位异或	^=	按位异或赋值
<<	左移	<<=	左移赋值
>>	右移	>>=	右移赋值
~	按位取反		

不存在 a~=b 的运算，这因为~是单目运算符，

位运算只能对整数进行运算

要区分&&和&

Cout<<(11&12);　结果为 8;

Cout<<11&12<<endl;　编译错误

Cout<<(11&&12);　结果为 1

Cout<<11&&12<<endl;　编译错误

Cout<<(11^12);　结果为 7

Cout<<11^12;　编译错误

Cout<<(16<<2)<<endl;　结果是 64　左移多少位就乘几个 2，对负数同样成立

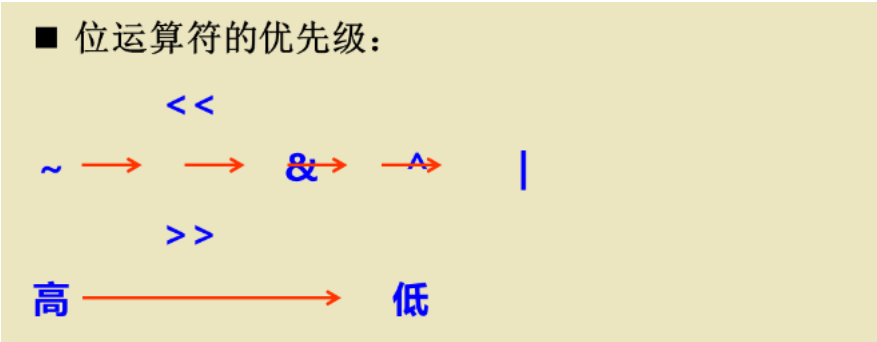
Cout<<16<<2<<endl;　结果是 162

Cout<<(16>>2)<<endl; 结果是 4，右移多少位就除以几个 2，对负数同样成立

Cout<<16>>2<<endl; 编译错误

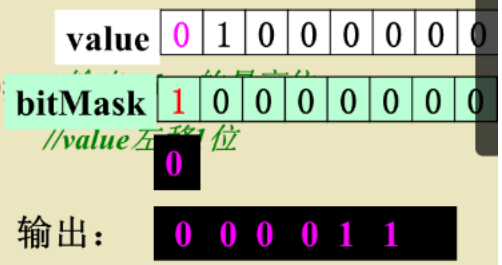
Cout<<!11　结果为 0

Cout<<!!11　结果为 1



一种利用按位与运算的输出二进制数的方法

```
void bitDisplay(unsigned value) //先考虑单个字节的情况
{ unsigned c;
  unsigned bitMask=1<<7;          //掩码，最高位置1
  cout<<value<<'=';
  for(c=1;c<=8;c++)
  { cout<<(value&bitMask?'1':'0');
    value<<=1;                    //value左移1位
    if(c%8==0)   cout<<' ';
  }
  cout<<endl;
}
```

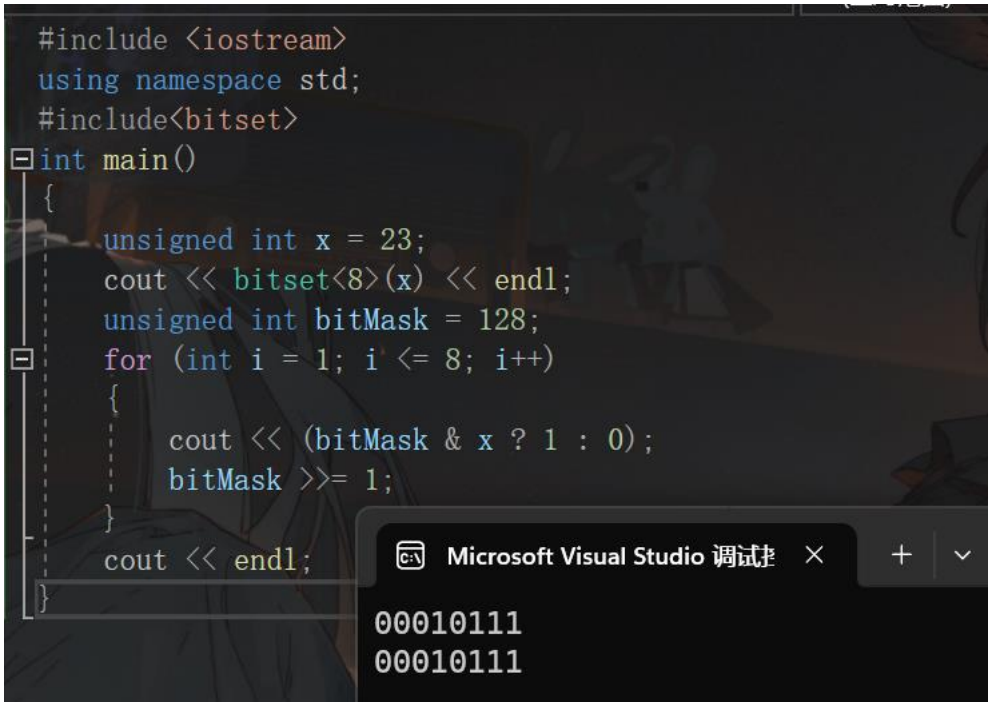


这样输出的是 8 个字节

可以修改成 32 字节的输出

```
void bitDisplay( unsigned value )
{ unsigned c;
  unsigned bitMask=1<<31;
  for( c=1; c<=32; c++ )
    { cout << ( value&bitMask ? '1' : '0' );
      value <<= 1;
      if( c%8 == 0 )
        cout << ' ' ;
    }
  cout << endl ;
}
```

这里不建议循环用从 0~31，不然输出‘ ’的时候会有问题



实际上也可以通过掩码左移实现

Bitset 函数可以输出二进制格式（在头文件<bitset>中）

异或的性质及运用

性质

1、交换律（即a ^ b == b ^ a）

2、结合律（即(a ^ b) ^ c == a ^ (b ^ c)）

3、对于任何数x，都有x ^ x == 0，x ^ 0 == x

4、自反性 A ^ B ^ B == A ^ 0 == A

即对给定的数A，用同样的运算因子（B）作两次异或运算后仍得到A本身。

这是一个神奇的性质，利用这个性质，可以获得许多有趣的应用。

一种基于异或运算的数据恢复（加密解密）

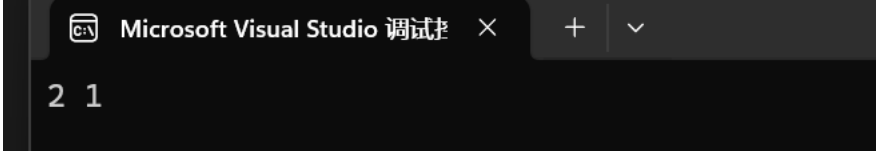
```
#include<iostream>
using namespace std;
int main()
{
   int a = 123, b = 456;
   cout << "a = " << a<< "\tb = " << b << endl;
   a = a ^ b;
   cout << "a = a ^ b\t" << a << endl;
   a = a ^ b;
   cout << "a = a ^ b\t" << a << endl;
   return 0;
}
```

一种基于异或运算的交换变量

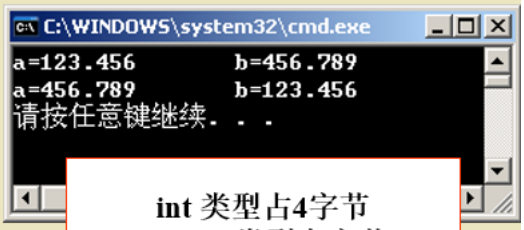
但如果使用异或，就可以节约一个变量的存储空间。
设有A、B两个变量，存储的值分别为a、b，则以下三行
式将互换它们的值。
表达式（值）：
A=A ^ B (a ^ b)
B=B ^ A (b ^ a ^ b = a)
A=A ^ B (a ^ b ^ a = b)

```
#include <iostream>
using namespace std;
int arr[2][2] = { 1,1,1,1 };

int main()
{
    int a = 1, b = 2;
    a = a ^ b, b = a ^ b, a = a ^ b;
    cout << a << " " << b << endl;
}
```



```
#include<iostream>
using namespace std;
int main()
{   double a=123.456, b=456.789;
    int*ap,*bp;
    ap = (int*)(&a);
    bp = (int*)(&b);
    cout<<"a="<<a<<"\tb="<<b<<endl;
    *ap=(*ap)^*(bp);     *bp=(*ap)^(*bp);     *ap=(*ap)^(*bp);
    ap++;   bp++;
    *ap=(*ap)^*(bp);     *bp=(*ap)^(*bp);     *ap=(*ap)^(*bp);
    cout<<"a="<<a<<"\tb="<<b<<endl;
}
```



int 类型占4字节
double 类型占字节
分别处理两个字节数据

浮点型通过整形指针来交换数

【问题描述】数组中找出唯一的一个重复元素值。

将数字1～1000放在含有1001个元素的数组中，只有唯一的一个元素值重复，其它均只出现一次。每个数组元素只能访问一次，设计一个算法，将它找出来；不用辅助存储空间，能否设计一个算法实现？

请你给出解决思路并拍照上传（可以是算法描述、也可以是源代码）。

- **解法一：**显然已经有人提出了一个比较精彩的解法，先将所有数加起来，再减去1+2+...+1000的和。这个算法已经足够完美了，相信出题者的标准答案也就是这个算法，唯一的问题是，如果数据过大，则可能会导致溢出。

- **解法二：异或就没有这个问题，并且性能更好。**将所有的数全部异或，得到的结果与1^2^3^...^1000的结果进行异或，得到的结果就是重复数。**[自反性 A ^ B ^ B = A ^ 0 = A]**

5.2 集合

集合是不能精确定义的基本数学概念。一般认为，当一些事物是可以按照某种性质（属性）分辨，这种事物构成的整体就称为集合。

根据集合的属性，可以断定某个特定的事物是否属于这个集合。如果属于，就称它为这个集合的元素。

5.2.2 集合运算的实现

1、集合数据的表示

➢ **数据表示**

unsigned A;

集合	二进制位串值	十进制值
{ 1, 3, 6 }	00100101	37
{1, 2, 5, 7 }	01010011	101
{ 2, 3, 5, 7, 8 }	11010110	214

即一个整数可以表示一个集合

2、集合基本运算实现

参考书本（有误）

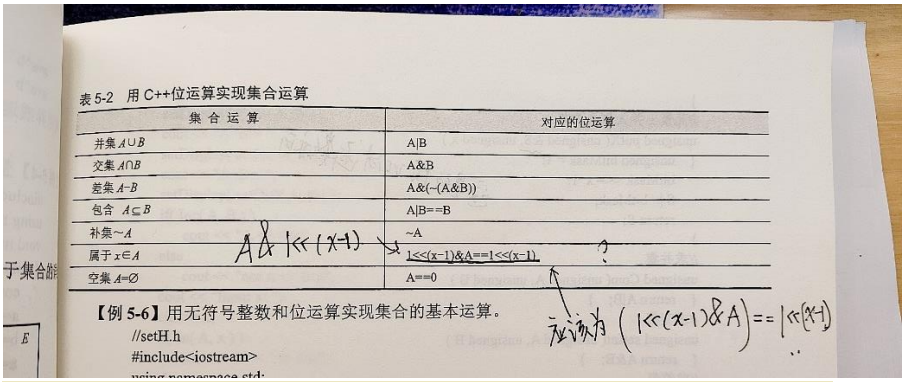


表 5-2　用 C++位运算实现集合运算

集 合 运 算	对应的位运算
并集 $A \cup B$	A|B
交集 $A \cap B$	A&B
差集 $A-B$	A&(~(A&B))
包含 $A \subseteq B$	A|B==B
补集 $\sim A$	~A
属于 $x \in A$	1<<(x-1)&A==1<<(x-1)
空集 $A=\varnothing$	A==0

【例 5-6】用无符号整数和位运算实现集合的基本运算。

```
//setH.h
#include<iostream>
```

➢ 基本运算实现

```
unsigned A, B;        //表示2个集合变量
unsigned x;           //表示集合元素
```

并集　$A \cup B$　　A|B

$A = \{1, 2, 5\}$

$B = \{2, 5, 7\}$

$A \cup B = \{1, 2, 5, 7\}$

A	0	0	0	1	0	0	1	1
B	0	1	0	1	0	0	1	0
A|B =	0	1	0	1	0	0	1	1

➢ 基本运算实现

```
unsigned A, B;        //表示2个集合变量
unsigned x;           //表示集合元素
```

交集　A∩B　　A&B

$A = \{1, 2, 5\}$

$B = \{2, 5, 7\}$

$A \cap B = \{2, 5\}$

A	0	0	0	1	0	0	1	1
B	0	1	0	1	0	0	1	0
A&B =	0	0	0	1	0	0	1	0

➢ 基本运算实现

```
unsigned A, B;      //表示2个集合变量
unsigned x;         //表示集合元素
```

差　A-B　　A&(~(A&B))

$A = \{1, 2, 5\}$
$B = \{2, 5, 7\}$
$A - B = \{1\}$

A	0	0	0	1	0	0	1	1
B	0	1	0	1	0	0	1	0
A&B	0	0	0	1	0	0	1	0
~(A&B)	1	1	1	0	1	1	0	1
A&(~(A&B))	0	0	0	0	0	0	0	1

差　A-B　　A&(~B)//可以吗？

➢ 基本运算实现

```
unsigned A, B;      //表示2个集合变量
unsigned x;         //表示集合元素
```

包含　A⊆ B　　A|B==B

$A = \{1, 2\}$
$B = \{1, 2, 5, 7\}$
$A \subseteq B$ 为真

A	0	0	0	0	0	0	1	1
B	0	1	0	1	0	0	1	1
A|B	0	1	0	1	0	0	1	1

A|B==B　等于 true

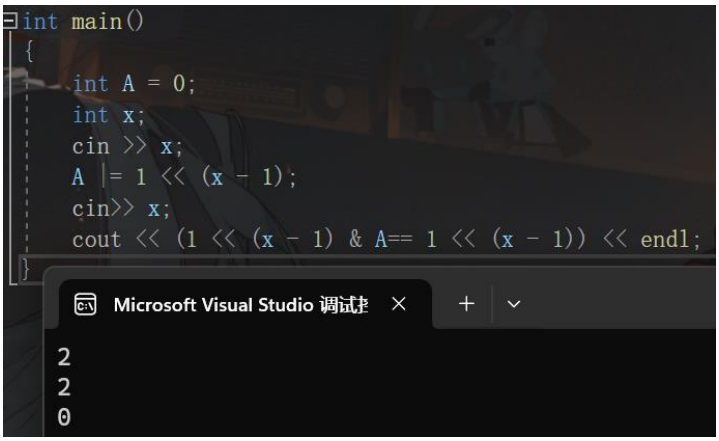
➢ 基本运算实现

```
unsigned A, B;      //表示2个集合变量
unsigned x;         //表示集合元素
```

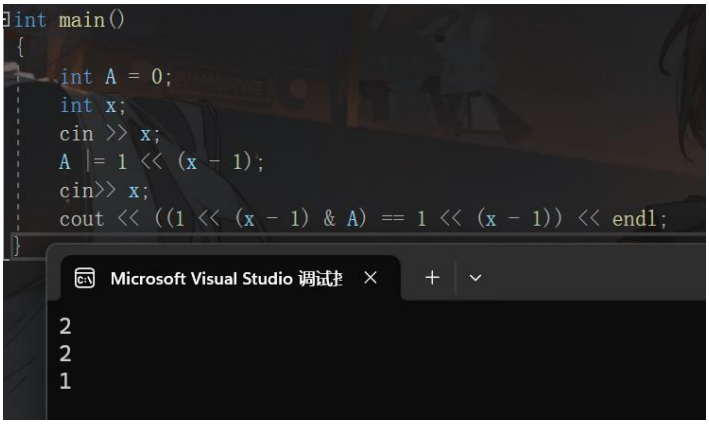
属于　x∈A　　1<<(x-1)&A == 1<<(x-1)

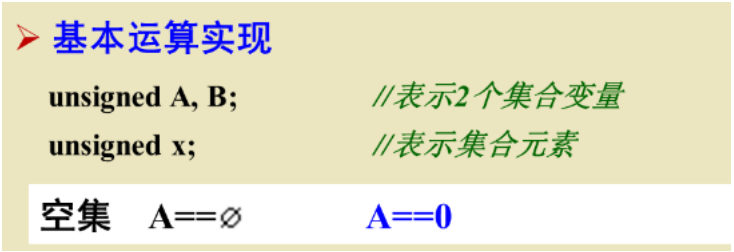
有误！应该是（1<<(x-1)&A）==1<<(x-1)或者 A&1<<(x-1)注意运算优先级

误：



正：





具体实践：课本 P141-143；得到 32n 个元素数组的方法 P143-144

但这两个例题的函数参数或许有点问题，了解思路

5.3 结构

5.3 结构

- 结构由数目固定的成员构成
- 各成员可以具有不同的数据类型
- 一个结构变量在内存占有一片连续的存储空间

5.3.1 定义结构

结构类型定义形式为：

```
struct 标识符
{ 类型 成员1;
  类型 成员2;
  ...
  类型 成员n;
};
```

例：

```
struct employee
{ char name [ 10 ];
  long code;
  double salary;
  char address [ 50 ];
  char phone [ 20 ];
};
```

5.3.1 定义结构

说明

(2) 一个结构类型的成员
可以是另一个已定义的结构类型

例如:
为职工结构添加出生日期信息
类型和变量声明为:

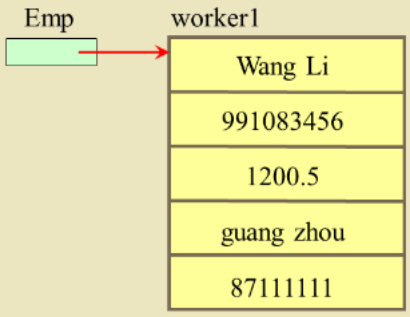
```
struct date
{ int month ;
  int day ;
  int year ;
} ;
struct employee
{ char name [ 10 ] ;
  date birthday ;
  long code ;
  double salary ;
  char address [ 50 ] ;
  char phone [ 11 ] ;
} worker1, worker2 ;
```

但是不能自己递归!

5.3.1 定义结构

说明

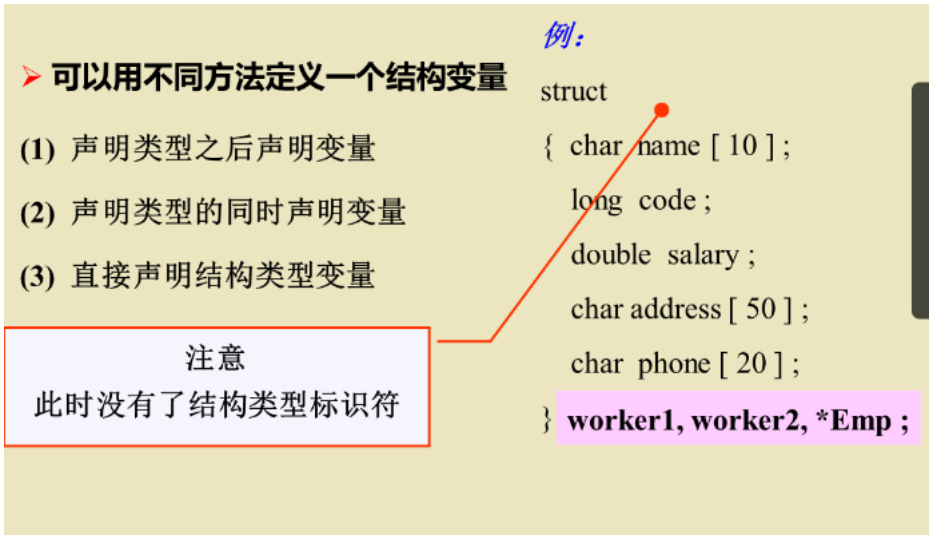
(1) 结构变量占有一片连续内存空间，具有结构类型的特征

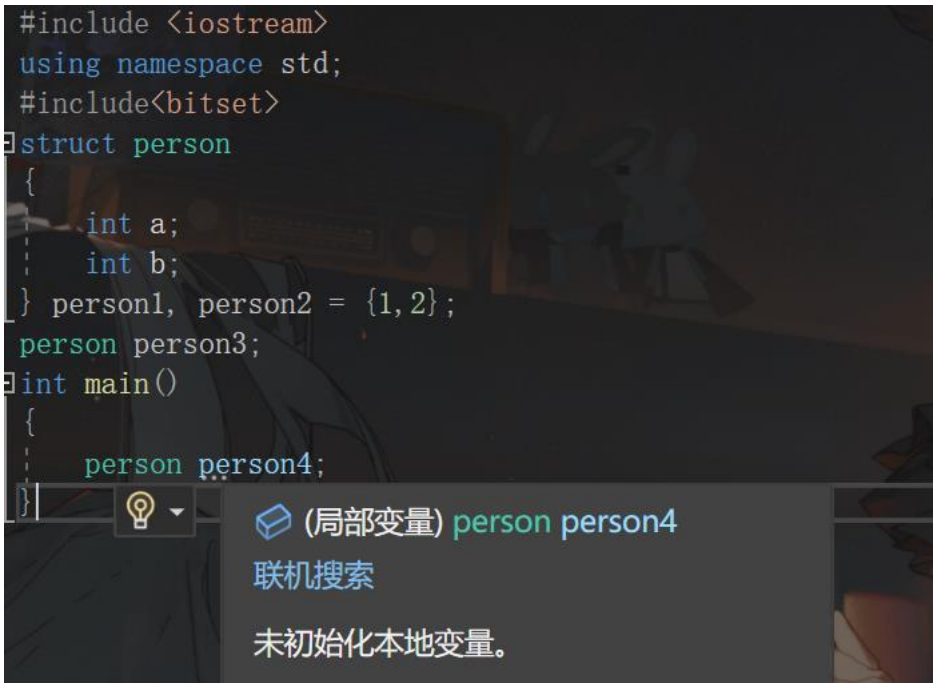


例:

```
struct employee
{ char name [ 10 ] ;
  long code ;
  double salary ;
  char address [ 50 ] ;
  char phone [ 20 ] ;
} ;
employee worker1, worker2,
         *Emp = &worker1 ;
```

注意结构中的成员和普通变量不一样在于：普通变量直接开辟空间，而结构变量是先给结构变量开辟空间再给结构变量中的每个成员开辟空间

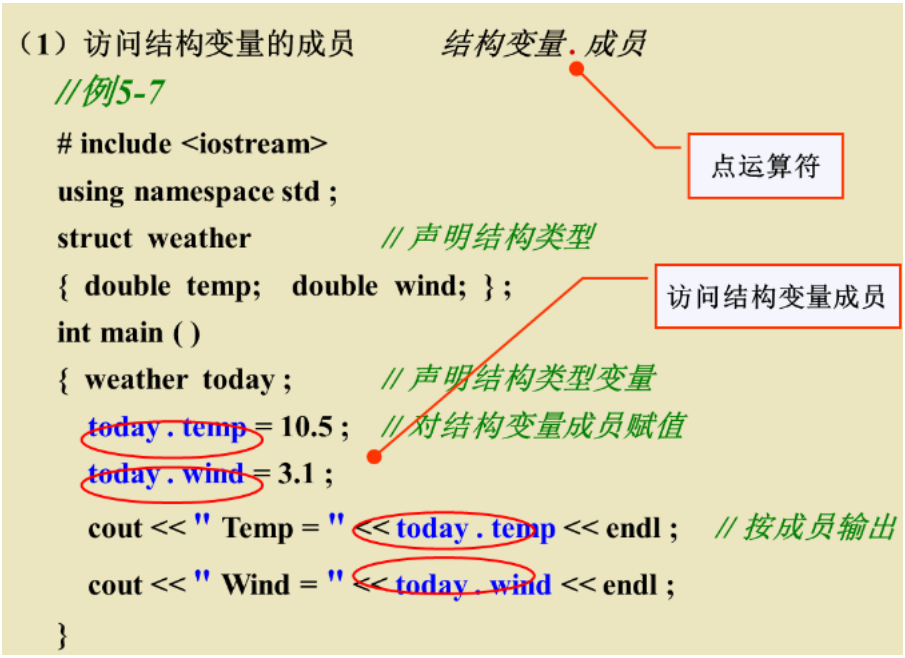




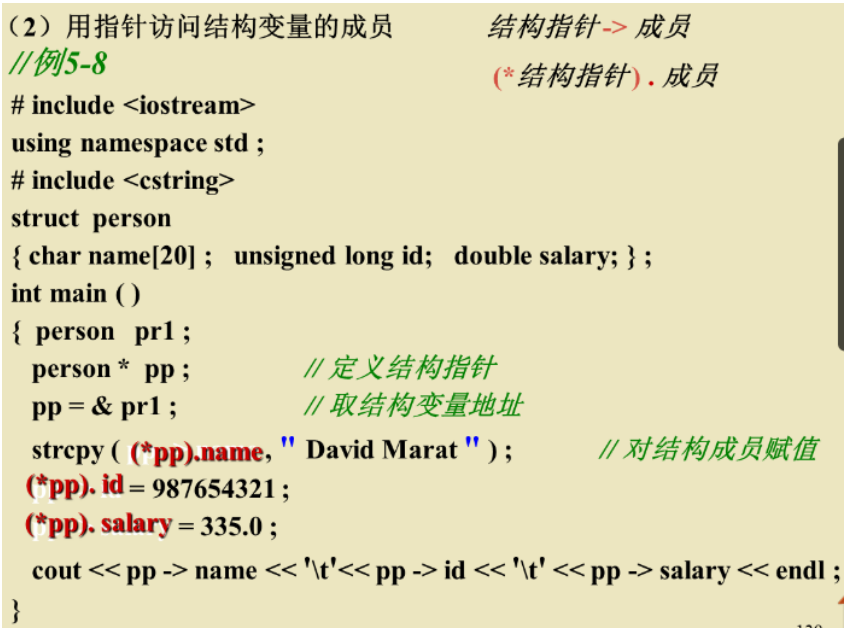
几种定义方式略有不同，可以在定义的时候初始化

5.3.2 结构访问

（1）直接访问（针对结构变量）



（2）以指针方式访问（针对结构指针）



这里可以不用 strcpy 改成 char *name;　pp->name={David Marat};

（3）类型相同的结构变量可以整体赋值


```
//例5-9
# include <iostream>
using namespace std ;
struct  weather
{ double  temp;   double  wind; } yesterday ;
int main ( )
{ weather  today ;
  yesterday . temp = 10.5 ;
  yesterday . wind = 3.1 ;
  today = yesterday ;        // 结构变量整体赋值
  cout << " Temp = " << today . temp << endl ;
  cout << " Wind = " << today . wind << endl ;
}
```

145

结构类型相同才可以整体赋值，如果两个结构类型不都是 weather 那不能整体赋值

“类型相同的变量” 是指用同一类型标识符说明的变量

例如：

```
struct  weather1
{ double  temp;   double  wind; } yesterday ;
struct  weather2
{ double  temp;   double  wind; } today ;
```

yesterday 和 today
尽管成员相同，但不是同类型变量
不可以整体赋值

5.3.3 结构参数

通过结构方式，依据某个元素对几个相同结构进行排序，做到整体排序

例5-10 对结构数组以某一成员作关键字排序

```
# include <iostream>
using namespace std ;
struct  person                  // 结构定义
{  char  name[10] ;  unsigned  int  id;  double  salary ;  } ;
person  allone[6] ;             // 结构数组声明
int main ( )
{  int  i ;   person  temp ;                  // 结构变量声明
   for ( i = 0 ;  i < 6 ;  i ++ )             // 输入数据
     { cout << i << ": name: " ; cin >> allone[i].name  ;
       cout << "id: " ;     cin >> allone[i].id ;
       cout << "salary: " ;     cin >> allone[i].salary ;
       cout << endl ;
     } ;
cout << "Sort:\n" ;
  for  ( i=1 ;  i < 6 ;  i ++ )              // 以成员salary作关键字排序
  {  for ( int  j = 0 ;  j <= 5-i ;  j ++ )
     { if  ( allone[j].salary > allone[j+1].salary  )    // 结构变量的整体交换
          {  temp = allone[j] ;   allone[j] = allone[j+1] ;    allone[j+1] = temp ;  }
     }
  }
  for ( i = 0 ;  i < 6 ;  i ++ )                                  // 输出排序后数据
     cout << allone[i].name << '\t' << allone[i].id << '\t' << allone[i].salary << endl ;
}
```

本质还是基本的排序算法

讨论

问题： 结构变量的整体交换降低了排序效率

解决： 使用索引机制，建立结构指针数组

方法：

1. 建立索引数组
2. 以关键字作依据进行数据比较，移动索引
3. 通过索引访问数据

应用索引数组可以通过关键字比较内容，减少结构变量的大量交换，减轻程序压力

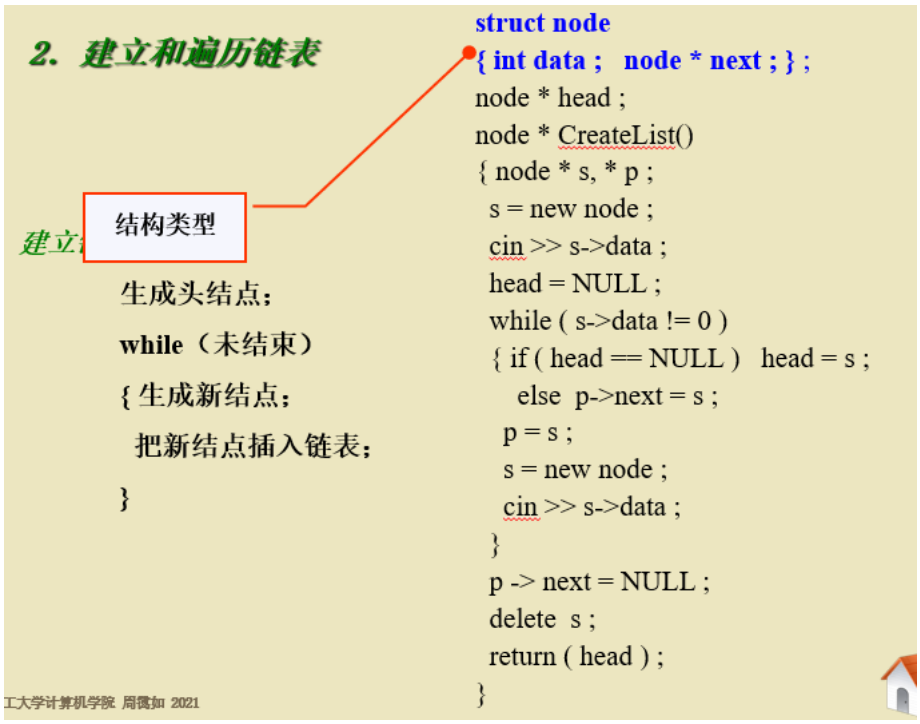
具体措施：

1、构造一个数组存放结构数组中各个变量的地址然后排序交换地址

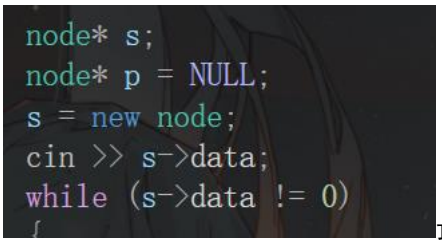
2、构造一个数组存放结构数组中各个变量的下表通过 allone[pa[i]]的方式比较然后交换下表（p[i]）

5.5 链表

- **程序对数据的表示，不但要求存放基本信息，还要表示与其它数据元素的关系**
- **线性表是最简单的数据组织形式**

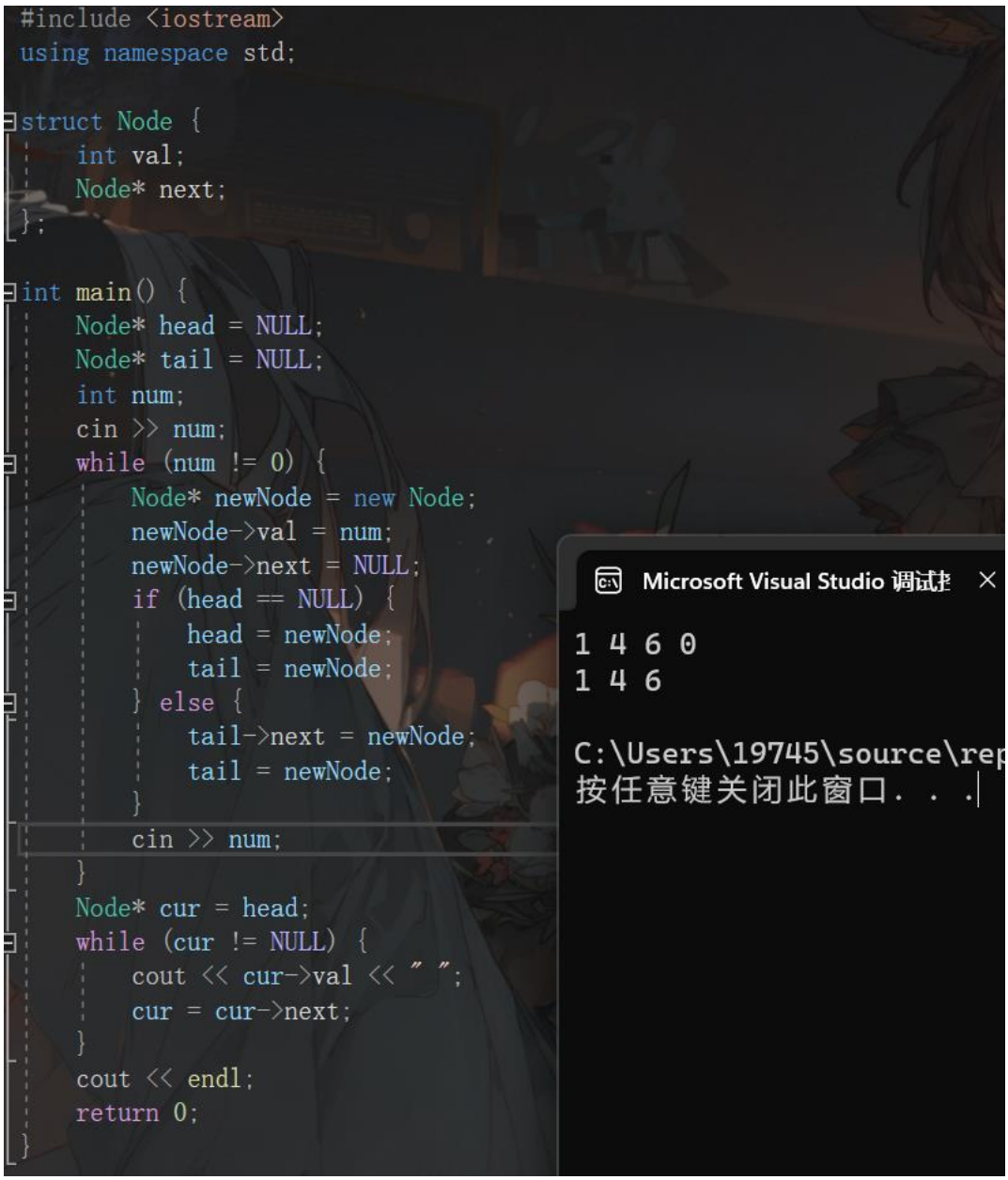


书本 P156 写法可能报错，



最好在定义*p 时候赋空值

比较好的一种写法（推荐）



```cpp
#include <iostream>
using namespace std;

struct Node {
    int val;
    Node* next;
};

int main() {
    Node* head = NULL;
    Node* tail = NULL;
    int num;
    cin >> num;
    while (num != 0) {
        Node* newNode = new Node;
        newNode->val = num;
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
        cin >> num;
    }
    Node* cur = head;
    while (cur != NULL) {
        cout << cur->val << " ";
        cur = cur->next;
    }
    cout << endl;
    return 0;
}
```

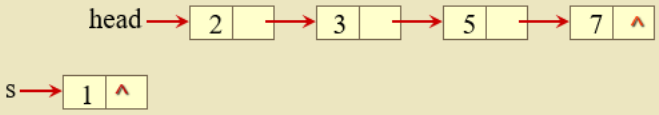
```
Microsoft Visual Studio 调试控
1 4 6 0
1 4 6

C:\Users\19745\source\rep
按任意键关闭此窗口. . .
```

3. 插入结点

- 在表头插入结点

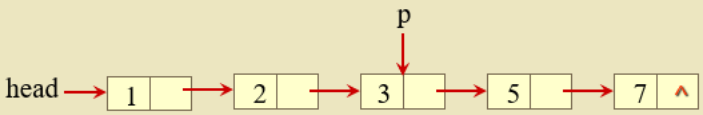
```
s -> next = head ;
head  = s ;
```



两条语句不能调换，这样会导致一直循环

- 在*p之后插入*s

```
s -> next = p -> next ;
p -> next  = s ;
```

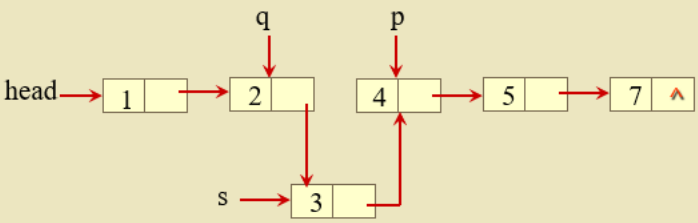


两条语句不能调换，会导致后面丢失并且一直循环

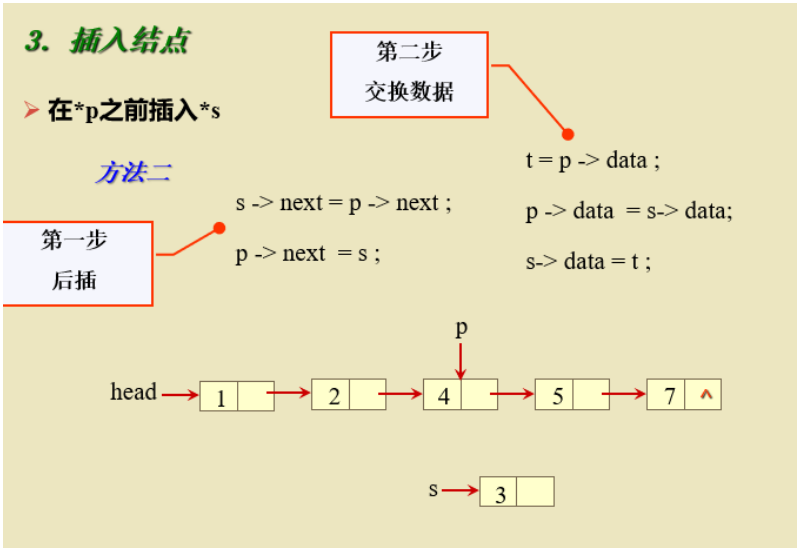
- 在*p之前插入*s

方法一

```
s -> next = p ;
q -> next  = s ;
```



利用辅助指针，可以做到插入多个节点



这种方法仅仅适合插入一个节点

应用：生成一个有序列表

```
if (表空) 生成链表的第一个结点；
else if ( num < head->data )
        把 num 插入头结点之前；          //此时num是最小值
    else
        { if (找到) 找第一个大于num的结点*p；
          把num插入*p之前；
        }
      else 把num插入表尾；       //此时num是最大值
```

例5-12 用插入法生成一个有序链表

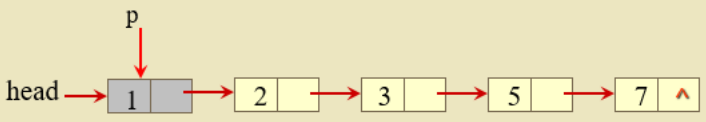
```
#include<iostream>
using namespace std ;
struct list
{ int data ;  list * next ; } ;
list * head ;
list * insert ( int num )
{ list * s, *p, *q ;
 s = new list ;
 s->data = num ;  s->next = NULL ;
 if ( head == NULL )
   { head = s ;   return( head ) ; }
 if ( head->data > s->data )
  { s->next = head ;   head = s ;
    return ( head ) ;
  }
 for ( q = head, p = head->next ;  p ; q = p, p = p->next )
  if ( p->data > s->data )
   { s->next = p ;      q->next = s ;
    return ( head ) ;
   }
 q->next = s ;
 return ( head ) ;
}
```

```
void showlist( const list * head )
{ cout << "now the items of list are: \n" ;
  while( head )
  { cout << head->data << '\t';    head = head->next ;  }
  cout << endl ;
}
 int main()
{ int k ;
  head = NULL ;
  cin >> k ;
  while( k != 0 )
    { head = insert(k) ;    cin >> k ; }
  showlist( head ) ;
}
```

PPT 中展示的是书本 P158-159 的程序，此外还有很多种思路生成有序链表（见 word）文件

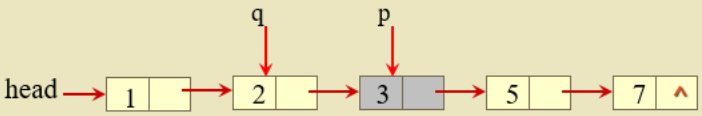
删除头结点

```
p = head ;
head  = head->next ;
delete p ;
```



➢ **删除结点 *p**

```
q->next  = p->next ;
delete p ;
```



这里运用辅助指针会方便很多（双指针在查找删除中的会比较方便）

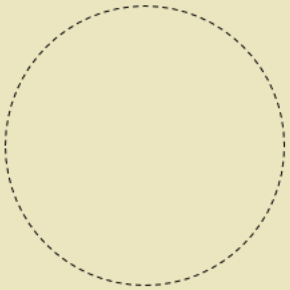
应用

例5-13　约瑟夫（Jonsephus）问题

➢ **n个人围成一个环，从第i个开始，由1至interval不断报数，凡报到interval的出列，直到环空为止。**

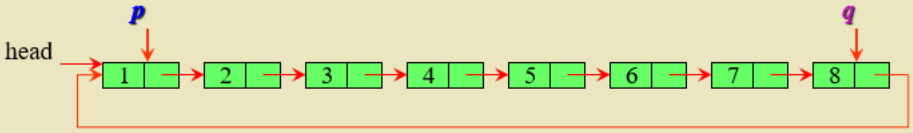
➢ **n=8，i=1，interval=3，则输出序列为：**

3　6　1　5　2　8　4　7



例5-13　约瑟夫（Jonsephus）问题

➢ 数据表示　　开始位置=2



➢ 对数据操作

```
找开始报数位置;
while ( 表结点数>1 )
{ for ( 1 To interval )
    { 结点计数;
      输出，删除第 interval 个结点 ;
    }
}
输出，删除最后一个结点 ;
```

具体程序可见书本 P160-161