实验舱蛟龙三班

指针 链表 及复习

zlj

2022.12

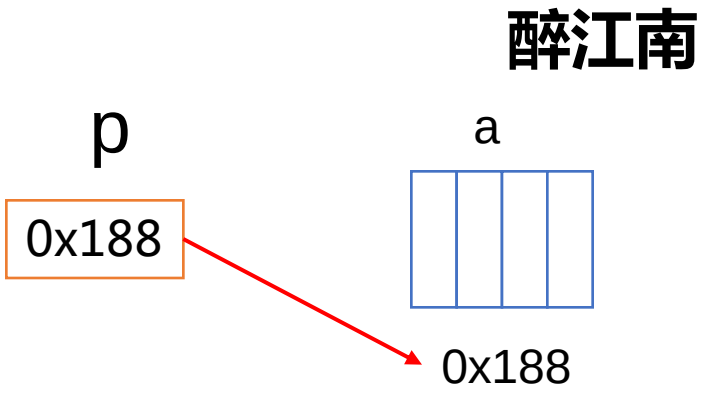
一、什么是指针？

在计算机科学中，指针（Pointer）是编程语言中的一个对象，利用地址，它的值直接指向（points to）存在电脑存储器中另一个地方的值。[1]

指针：也就是内存地址

我们可以把内存的每个字节都想像成饭店的一个包间，那么**内存地址**相当于房间号（以字节为单位）。

变量名就相当于给地址起一个好记的名称。



指针概念

指针，就是内存地址；

指针变量　**存放**内存地址的特殊变量。

通过指针，能够对该指针指向的内存区域进行读写。

指针变量定义

类型名 * 指针变量名;

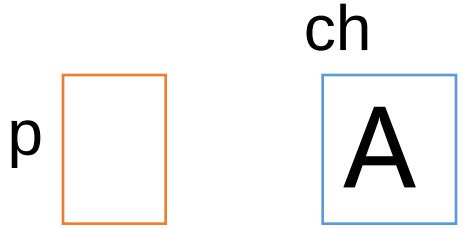
例:

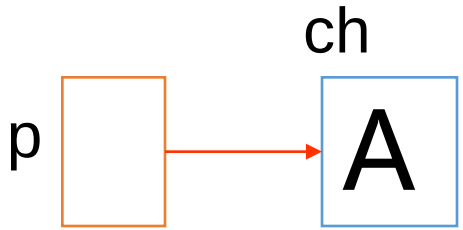
```
int * p;      //变量p是一个指针，p指向的内存空间是int类型
char * pc;    //变量pc是一个指针,pc指向的内存空间是char类型
float *pf;    //变量pf是一个指针，pf指向的内存空间是float类型
```

指针变量的操作

- 给指针变量赋值

```
int *p=NULL;
char ch='A';
p=&ch;
```





指针变量的基本操作：

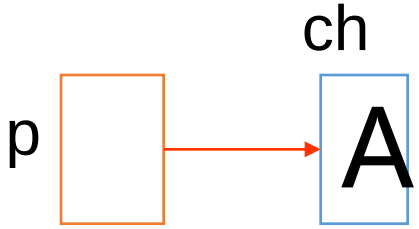
操作	样例
定义指针	int *p=NULL; 或 float *p1;
取地址运算符： &	int a=10; p=&a;
间接运算符： *	*p=10;
指针变量直接存取是内存地址	cout<<p; 结果可能是:0x444ce……
指针变量间接存取的：	cout<<*p; 结果是：10

* 在定义语句中只表示变量的类型是指针，没有任何计算意义。* 在语句中表示“指向”运算。&表示取“地址”。

指针变量的操作

- 给指针变量赋值

```
int *p=NULL;
char ch='A';
p=&ch;
```



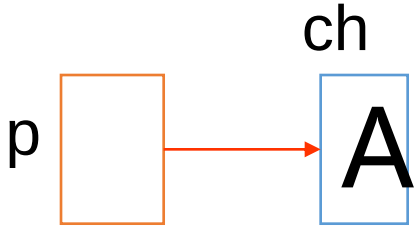
- 给指针变量指向的内存空间赋值

```
int a;
int *pc;
pc=&a;
*pc=56;
```

指针变量的操作

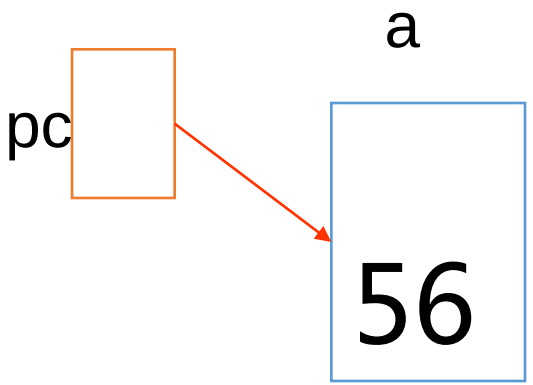
- 给指针变量赋值

```
int *p=NULL;
char ch='A';
p=&ch;
```



- 给指针变量指向的内存空间赋值

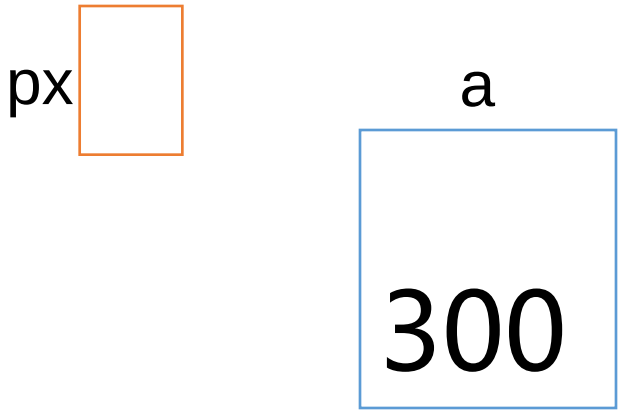
```
int a;
int *pc;
pc=&a;
*pc=56;
```



指针变量的操作

- 指针变量互相赋值

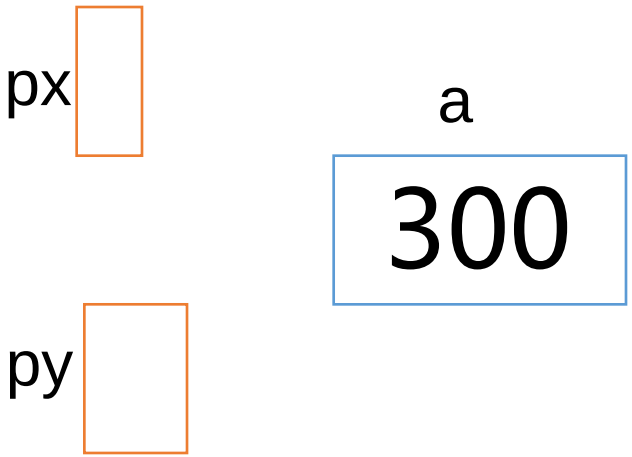
```
int a=300;
int *px=&a;
int *py;
py=px;
```



指针变量的操作

- 指针变量互相赋值

```
int a=300;
int *px=&a;
int *py;
py=px;
```



指针变量的操作

- 指针变量互相赋值

```
int a=300;
int *px=&a;
int *py;
py=px;
```

- 指针变量的++或--运算

```
int a=86;
int *p=&a;
p++;
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int *p;
    int a=5;
    p=&a;
    cout <<a<<endl;
    cout <<p<<endl;
    cout <<*p<<endl;
    cout <<++*p<<endl;
    cout <<++p<<endl;
    cout <<*p<<endl;
    return 0;
}
```

```
5
0x70fe44
5
6
0x70fe48
7405128

--------------------

Process exited after 0
请按任意键继续. . .
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int a=300;
    int *px=&a;
    int *py;
    py=px;
    cout <<a<<endl;
    cout <<px<<endl;
    cout <<py<<endl;
    cout <<*px<<endl;
    cout <<*py<<endl;

    return 0;
}
```

指针 在程序中的作用

- 有了指针，就有了自由访问内存空间的手段
- 不需要通过变量，就能对内存直接进行操作。通过指针，程序能访问的内存区域就不仅限于变量所占据的数据区域
- 在C++中，用指针p指向a的地址,然后对p进行加减操作，p就能指向a 后面或前面的内存区域，通过p也就能访问这些内存区域

指针在函数中的应用

```
3  void swap(int *x,int *y){
4      int tem=*x;
5      *x=*y;
6      *y=tem;
7  }
8  int main() {
9      int a,b;
10     cin>>a>>b;
11     swap(&a,&b);
12     cout<<a<<" "<<b<<endl;
13     return 0;
```

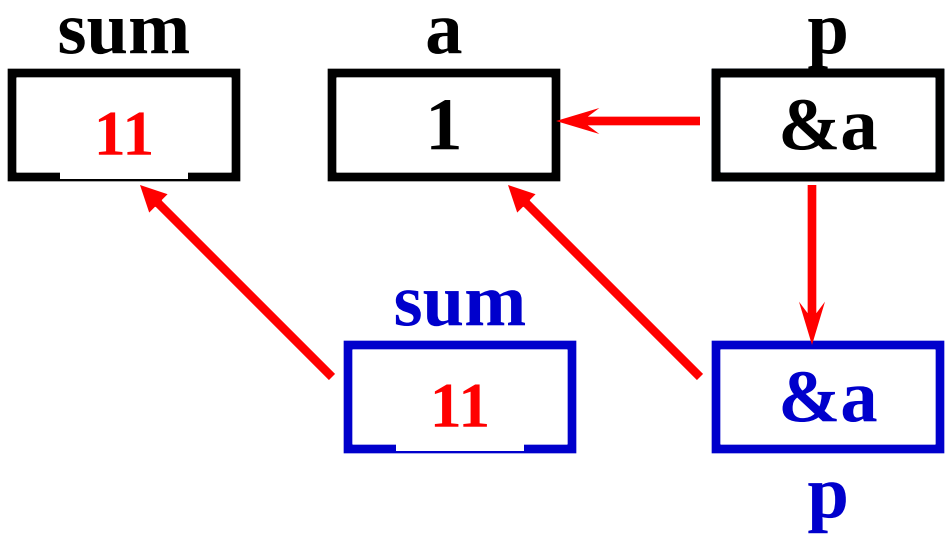
变量的引用：

```
3  void Swap(int &x,int &y){
4      int tem=x;
5      x=y;
6      y=tem;
7  }
8  int main() {
9      int a,b;
10     cin>>a>>b;
11     Swap(a,b);
12     cout<<a<<" "<<b<<endl;
```

阅读程序：

```
int s( int  *p)
{    int  sum=10;
     sum=sum + *p;
     return sum;
}
int main( )
{  int  a=0, i, *p, sum;
   for (i=0; i<=2; i++)
     {   p=&a;
         cin>>*p;
         sum=s(p);
       cout<<“sum=”<<sum<<endl;
     }
}
```

输入：1 3 5<CR>



sum=11

sum=13

sum=15

指针与数组

- **数组名就是一个指针常量，它指向数组的起始地址。**

```
int a[100];
int *p;
p=a;
for (int i=0;i<n;i++) cin >>*(a+i);
for (int i=0;i<n;i++) cout <<*(p+i);
```

-

动态数组

- 指针申请空间

```
int *p=new(int);
```

- 指针申请数组空间

```
int n;
int *a;
cin >>n;
a=new int [n+1];
```

三、自引用结构体

1、定义：当一个结构体中有一个或多个成员是指针，它们指向的类型就是本结构体类型。就叫引用自身的结构体。

例：

```
struct  student{
        int x,y;
        student  *next;
       }stu;
```

这种结构是实现链表、动态数据结构的基石。

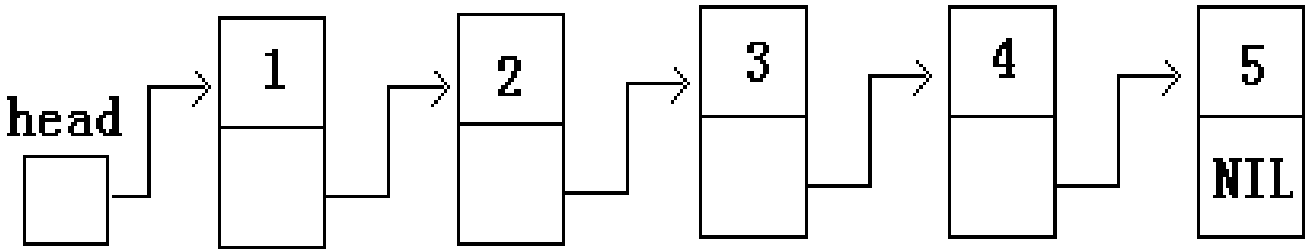
2、动态申请指针变量：

p=new student;　//动态申请，空间大小由指针变量的基类型决定。

free(p)

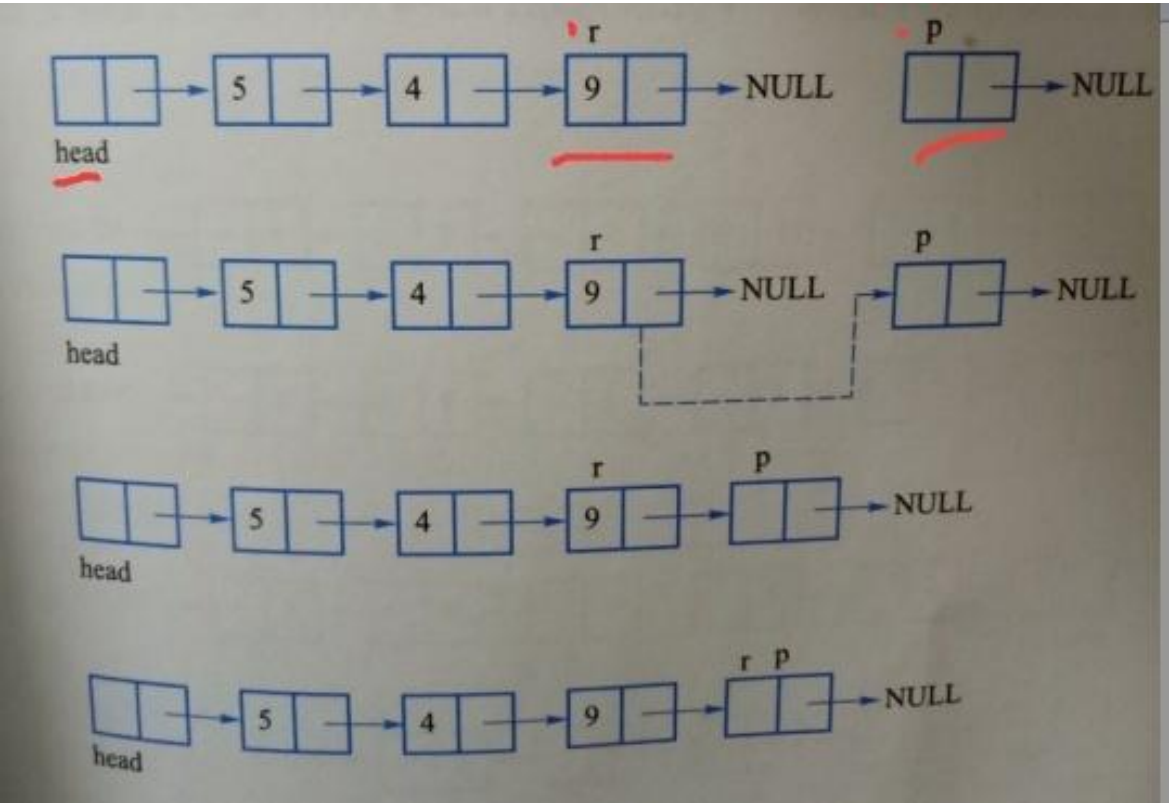
四、单链表的建立、输出

1、单链表的结构：



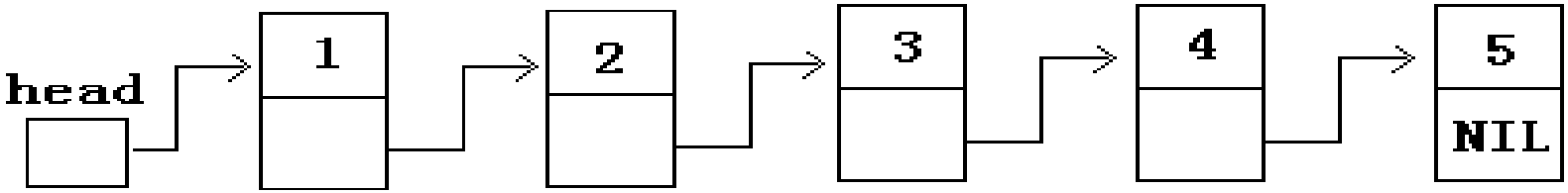

```
struct  node{
      int data;
      node  *next;
     };
node *head,*p,*r;
```

2、建立单链表（尾插法）



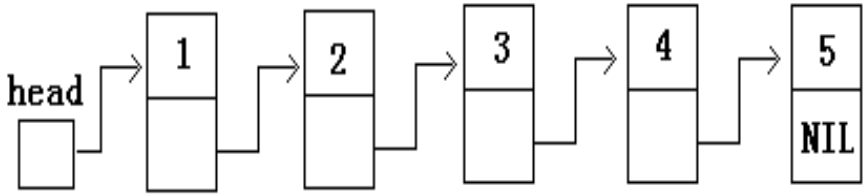

```
node *head,*p,*r; int x;
void read(){
      head=new node;
      r=head; cin>>x;
 while(x>0){
   p=new node;
   p->data=x;
   p->next=NULL;
   r->next=p;
   r=p;
   cin>>x;
   }
 }
```

3、输出单链表：



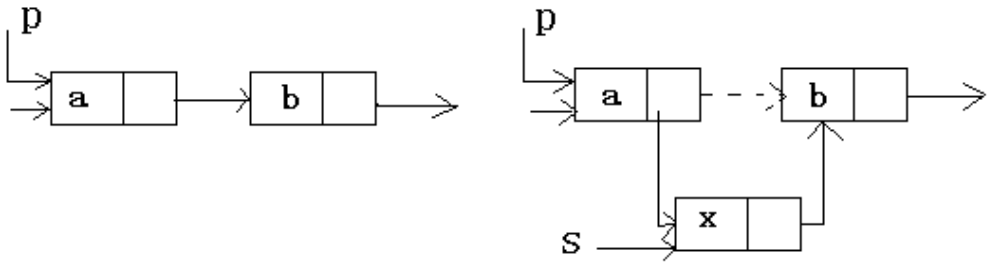

```
void print(){
 p=head->next;
while(p->next!=NULL){
 cout<<p->data<<” “;
 p=p->next;
 }
 cout<<p->data<<endl;
```

4、取单链表第i个结点的数据域：




```
void get(node *head,int i){
 node *p;int j;
p=head->next;j=1;
while((p!=NULL)&&(j<i)){
  p=p->next;j++;}
 if((p!=NULL)&&(j==i))
   cout<<p->data;
   else cout<<"i not";
}
```

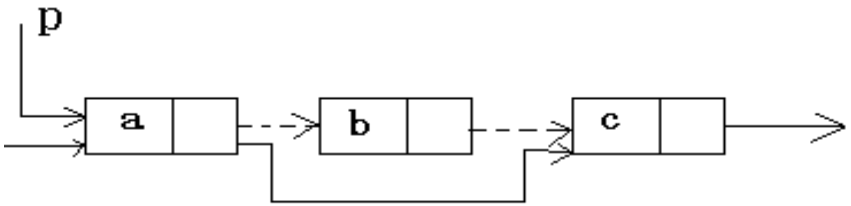
5、插入一个结点到链表：



插入结点前和后的链表变化

```
void insert(Node *head,int i,int x)//插入 x 到第 i 个元素之前
{ Node *p,*s;int j; p=head;
   j=0;
while((p!=NULL)&&(j<i-1))  //寻找第 i-1 个结点,插在它的后面
{ p=p->next; j=j+1;  }
if(p==NULL)cout<<"no this position!";
else{       //插入
s=new Node; s->data=x; s->next=p->next; p->next=s;}
}
```

6、删除第i个结点：



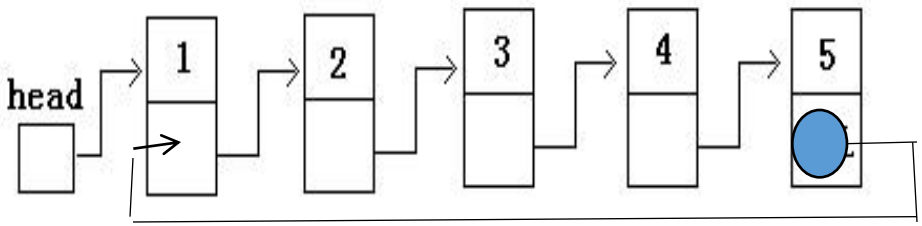
删除一个结点前和后链表的变化

```
void  delete(Node *head,int i)     //删除第 i 个元素
{  Node *p,*s;int j; p=head;j=0;
    while((p->next!=NULL)&&(j<i-1))
        {p=p->next; j=j+1;}  //p 指向第 i-1 个结点
   if (p->next==NULL)cout<<"no this position!";
     else {     //删除 p 的后继结点，假设为 s
        p->next=p->next->next;
     }
}
```

7、求单链表长度：

```
int len(Node *head)
{ int n=0; p=head; while(p!=NULL)
{
   n=n+1; p=p->next;
}
return n;
}
```

8、循环单链表：




```
void huan()
{head=new node;head->next=NULL;
r=head;cin>>x;
 for(int i=1;i<=n;i++){
   p=new node;
   p->data=x;p->next=NULL;
   r->next=p;r=p;
   cin>>x;
}
 r->next=head;r=head;
}
```

例2：链表插入数据

输入n个正整数(0<n<100000)，在第m位后插入一个整数x的值。(1<m<=10000)（请用链表来完成本题）

【输入说明】

两行，第一行3个整数，表示n,m,x的值。 第二行，n个整数，用空格分隔；

【输出说明】

一行，n+1个整数，表示插入数据后的链表序列。

【输入样例】

5 4 99

2 3 1 5 4

【输出样例】

2 3 1 5 99 4

```cpp
using namespace std;
struct Node {
    int data;
    Node *next;
};
Node *head,*p,*r,*k;
int n,m,x,y;
void in() { //建立链表
    cin>>n>>m>>y;
    head=new Node;
    r=head;
    for(int i=1; i<=n; i++) {
        cin>>x;
        p=new Node;
        p->data=x;
        p->next=NULL;
        r->next=p;
        r=p;
    }
}
```

```cpp
void insert(){ //链表插入节点
    int j=1;
    Node *s,*q;
    s=head;
    while(j<=m) {
        s=s->next;
        j++;
    }
    q=new Node;q->data=y;q->next=s->next;s->next =q;
}
void out() { //输出链表
    k=head->next;
    while(k->next!=NULL) {
        cout<<k->data<<" ";
        k=k->next;
    }
    cout<<k->data<<endl;
}
int main() {
    in();insert();out();
    return 0;
}
```

例3：链表删除数据

输入n个正整数(0<n<100000)，删除数列中第x位上的值。
(1<x<=100000)（请用链表来完成本题）

【输入说明】

两行，第一行2个整数，表示n,x的值。 第二行，n个整数，用空格分隔
;

【输出说明】

一行，n-1个整数，表示删除后的序列。

【输入样例】

5 4
2 3 1 5 4

【输出样例】

2 3 1 4

```
struct Node {
    int data;
    Node *next;
};
Node *head,*p,*r,*k;
int n,m,x,y;
void in() { //建立链表
    cin>>n>>m;
    head=new Node;
    r=head;
    for(int i=1; i<=n; i++) {
        cin>>x;
        p=new Node;
        p->data=x;
        p->next=NULL;
        r->next=p;
        r=p;
    }
}
```

```
void del() {  //链表删除节点
    int j=1;
    Node *s;
    s=head;
    while(j<m) {
        s=s->next;
        j++;
    }
    s->next=s->next->next;
}
```

```
void out() { //输出链表
    k=head->next;
    while(k->next!=NULL) {
        cout<<k->data<<" ";
        k=k->next;
    }
    cout<<k->data<<endl;
}
int main() {
    in();del();out();
    return 0;
}
```

五、复习

让计算机重复执行语句：

1、循环　while　for

2、自定义函数

3、递归

Hermite多项式

递归求二进制

给你 n 个十进制整数（整数可能会很大，大到 2^{63} ，请你用递归方法将这些十进制数转换成二进制数。）

请你用递归方法将这些十进制数转换成二进制数。

输入

第一行，一个整数 n 。

接下来 n 行，有 n 个整数。

输出

n 行二进制数。

输入样例

```
3
6
15
4
```

输出样例

```
110
1111
100
```

阅读程序：《递归求二进制》

```
int main() { //张怀瑾
    int x,k=0;
    long long n;
    bool a[1000001];
    cin>>x;
    for(int i=1; i<=x; i++) {
        cin>>n;
        memset(a,0,sizeof(a));
        k=0;
        while(n>0) {
            a[k++]=n%2;
            n=n>>1;
        }
        for(int j=k-1; j>=0; j--)
            cout<<a[j];
        cout<<endl;
    }
    return 0;
}
```

阅读程序：《递归求二进制》

```
2  using namespace std; // 顾益维
3  void jz(long long x) {
4      if(x==0) return;
5      jz(x/2);
6      cout<<x%2;
7  }
8  int main() {
9      int n;
10     long long a;
11     cin>>n;
12     for(int i=1; i<=n; i++) {
13         cin>>a;
14         jz(a);
15         cout<<endl;
16     }
17     return 0;
```

将多个不同类型的数据“捆绑” 在一起

1、结构体：struct{}　　pair<int,string>

2、重载运算符

《求援争先》

STL 基本应用

Vector set map string