

多重循环



课程安排

- 1、多重循环结构
- 2、嵌套注意事项
- 3、多重循环实例
- 4、编程题目：2412九九乘法表
- 5、编程题目：2413图形输出1
- 6、编程题目：2414图形输出2

多重循环结构

一个循环体语句中又包含另一个循环语句，称为循环嵌套。

```
for (int i = 1; i <= n; i++) { //外层  
    循环  
        for (int j = i; j <= m; j++) { //内  
            层循环  
                cout << i << " " << j << endl;  
            }  
    }  
}
```

输出结果：

1 1

1 2

1 3

2 2

2 3

多重循环结构

```
for (int i = 1; i <= n; i++) { //外层  
    循环  
        for (int j = i; j <= m; j++) { //内  
            层循环  
                cout << i << " " << j <<  
                endl;  
            }  
    }
```

i=1	i=2	i=3	...
1 1	2 2	3 3	
1 2	2 3	3 4	
1 3	2 4	3 5	...
...	
1 m	2 m	3 m	

嵌套注意事项

书写规范：

- 1、循环嵌套结构采用“右缩进”格式
- 2、内层循环和外层循环的循环控制变量不能相同

主要应用：

C++语言中经常被用于按行列方式输出数据

```
for (int i = 1; i <= n; i++) { //外层循环
    for (int j = i; j <= m; j++) { //内层循环
        cout << i << " " << j << endl;
    }
}
```

多重循环实例

双重for循环	for和while循环混用	
<pre>for(int i = 0; i < 10; i++) { for(int j = 0; j <= i; j++) { cout << i * j << endl; } }</pre>	<pre>int b = 10; while(b > 0) { for(int a = 0; a < 10; a++) { cout << a + b << endl; } int c = 0; while(c < b) { cout << c + b << endl; c++; } b--; }</pre>	<pre>for(int a = 0; a < 10; a++) { int b = 10; while(b > 0) { cout << a + b << endl; b--; } }</pre>

多重循环实例

```
for(int a = 1; a <= 10;  
a++)  
{  
    for(int b = 0; b < a;  
b++)  
    {  
        cout << "**";  
    }  
    cout << endl;  
}
```

可以通过外层循环
的变量，控制内层
循环的循环次数。

输出结果：

```
*  
**  
***  
****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

多重循环实例

上面这段程序：第一行会输出1个 *。

在第一次循环时 $a=1$ ，因此内部循环会被执行1次。

之后会输出一个换行，因为在外部循环的后面，有一句输出换行：`cout << endl;`

2412 启蒙练习-九九乘法表

我们都学过九九乘法表，现在我们将它扩展成 12×12 的乘法表，请你输出这个乘法表的前N行。

输入

输入一个数N，其中 $1 \leq N \leq 12$

输出

输出一个乘法表，每行几个算式之间以空格隔开

5

$1*1=1$
 $1*2=2\ 2*2=4$
 $1*3=3\ 2*3=6\ 3*3=9$
 $1*4=4\ 2*4=8\ 3*4=12\ 4*4=16$
 $1*5=5\ 2*5=10\ 3*5=15\ 4*5=20$
 $5*5=25$

2412 九九乘法表（解题思路）

n的乘法表共有n行，因此考虑外层循环 $i=1 \rightarrow n$ ，并且在每次循环的最后部分输出一个换行。

第1行有1个算式，第2行有2个算式，……第*i*行有*i*个算式，因此内层循环 $j=1 \rightarrow i$

- 对于每一个算式，输出3个数， $j, i, j * i$ ，并配合相应的算式字符，到达要求的输出效果，注意算式间要输出空格。

2412 九九乘法表

参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) // 外循环控制
        输出行数
    {
        for (int j = 1; j <= i; j++) // 内循环控
            制输出列数
            cout << j << "*" << i << "=" << i * j
            << '\n'; // 输出乘积
        cout << endl; // 换行
    }
    return 0;
}
```

2413 启蒙练习-图形输出1

输入一个n，输出n层图形，见下（图为n=4的情况）：

*
**

输入

输入一个数n

输出

输出相应的图形

6

*
**

2413 图形输出1（解题思路）

双重循环：外层循环 $i=1->n$ ，内层循环 $j=1->i$



2413 图形输出1 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++)
            cout << "*";
        cout << endl;
    }
}
```

2414 启蒙练习-图形输出2

输入一个数n，输出n层图形，见下（图为n=4的情况）

*

输入

输入一个数n

输出

输出相应的图形

5

*

2414 图形输出2（解题思路）

双重循环：外层循环 $i=1 \rightarrow n$ ，内层循环 $j=1 \rightarrow 2i-1$



2414 图形输出2 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j < i * 2; j++)
            cout << "*";
        cout << endl;
    }
}
```

总结

我们一起学习了多重循环的知识点。

多重循环在我们以后的编程中会经常用到，三种嵌套的形式在NOIP的复赛编程中会经常考察。



多重循环的控制



课程安排

- 1、多重循环控制
- 2、编程题目：3211 数根
- 3、编程题目：2418 最大的质数
- 4、编程题目：2416 9的个数

嵌套中的break

内层循环的break，只让程序
跳出当前的内层循环
而外层的循环会继续执行。

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    for (int i = 2; i < 10; i++)
    {
        int a = 0;
        for (int j = 2; j < i; j++)
        {
            if (i % j == 0)
            {
                a = 1;
                break;
            }
        }
        if (!a)
            cout << i << " ";
    }
    return 0;
}
```

输出结果为：

2 3 5 7

3211 数根

将正整数 n 的各个位相加，得到一个新的数字 k ，如果这 k 是一位数，称 k 为 n 的数根。如果 k 不是一位数，则对其重复处理，直到 k 成为一位数，此时 k 也称为 n 的数根。

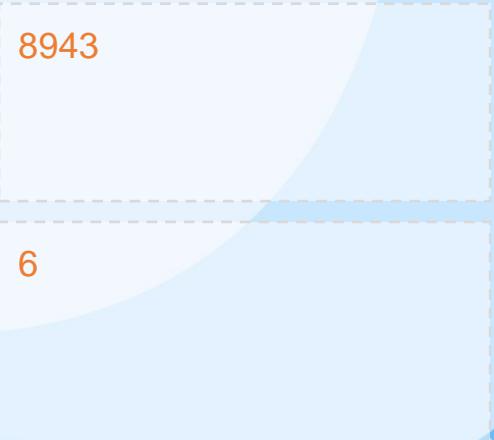
例如： $8943:8+9+4+3=24$ ， $24:2+4=6$ 所以 6 是 8943 的数根；
请编写程序，计算 n 的数根。

输入

- 输入一个正整数 n 。

输出

输出一个数，表示原数的数根。



8943

6

3211 数根（解题思路）

按照题目要求，计算数根，直到 $n \leq 9$ 为止。



3211 数根 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, cnt = 0;
    cin >> n;
    while (n > 9) {
        while (n > 0) {
            cnt += n % 10;
            n /= 10;
        }
        n = cnt;
        cnt = 0;
    }
    cout << n;
    return 0;
}
```

3211 数根（解题思路2）

本题还有一个巧妙的解法，我们判断一个数是否为 9 的倍数，可以将这个数各个数位求和，然后计算加和是否为 9 的倍数。这是因为 $10^n \bmod 9 = 1$ 。所以 x 对 9 求余数等于 x 的各个数位之和对 9 取余。

数位和对 9 取余余数不变，因此不论经过多少次数位求和，最终得到的 R 满足 $n \bmod 9 = R \bmod 9$ ，而 R 是一个小于等于 9 的数，所以如果 $n \bmod 9 = 0$ ，则 $R = 9$ ，否则 $R = n \bmod 9$

3211 数根 参考答案2

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    cout << ((n - 1) % 9 + 1) << endl;
    return 0;
}
```

2418 启蒙练习-最大的质数

读入一个数n，求比n小的最大的质数是谁。

一个数是质数，当且仅当除了1和它本身以外，没有其他的数能整除它。

输入

输入一个数n

输出

输出一个最大的质数

53

47

2418 最大的质数 解题思路

从 $n-1$ 向下枚举，逐个检测 每个数是否为质数，
如果是质数则直接返回。



2418最大的质数 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    for(int i = n - 1; i > 1; i--)
    {
        bool flag = false;
        for(int j = 2; j * j <= i; j++)
            if(i % j == 0)
                flag = true;
        if(!flag)
        {
            cout << i << endl;
            return 0;
        }
    }
    return 0;
}
```

2416 启蒙练习-9的个数

给个一个正整数n , 请你求出1到n之间所有整数中出现了多少个9

输入

输入一个正整数n , 其中 $1 \leq n \leq 100$

输出

输出9的个数

93

13

2416 9的个数 解题思路

我们要考虑的就是如何把 i 这个数字身上的 9 数出来，要解决这个问题我们就要——“提取”出 i 这个数字的每一位，然后再判断当前这一位是不是 9，如果是那么加一。

如何提取 i 这个数的每一位呢？我们简化问题，考虑如何提取一个数字 i 的最后一位，不难想到 i 这个数字的最后一位就是 $i \bmod 10$ ，那么倒数第二位呢，当然就是 $(i/10) \bmod 10$ ，这么看来只要我们用一层循环，每次让 i 这个数字整除 10，再提取最后一位，就能达到遍历 i 这个数字每一位的目的了。

2416 9的个数 参 考答案

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        int temp = i;
        while (temp) {
            if (temp % 10 == 9) ans++;
            temp /= 10;
        }
    }
    cout << ans << endl;
    return 0;
}
```

总结

我们一起学习了多重循环控制的知识点。

多重循环break控制在我们以后的编程中会经常用到，在NOIP的复赛编程中会经常考察。



一维数组



课程安排

- 1、什么是数组
- 2、一维数组
- 3、声明数组
- 4、数组的初始化
- 5、一维数组的使用
- 6、编程题目：2092 数组翻转
- 7、编程题目：1959 好事成双
- 8、编程题目：3386 纸杯猜数

思考一

1. 读入5个整数并保存到变量a,b,c,d,e中怎么写？

cin >> a >> b >> c >> d >> e ;

如果我们将对变量的命名加以调整，使得看起来更舒服有什么写法？

cin >> a0 >> a1 >> a2 >> a3 >> a4 ;

那么如何读入100个整数并把值保存到变量中，要怎么办呢？

思考二

我们已经学习过使用循环来读入多个数字

```
for(int i = 0; i < 10 ; i++)  
    cin >> n;
```

但如何把每一个n的值都保存下来？**这需要数组来帮忙**

引言：什么是数组

具有相同类型的若干元素按顺序的形式组织起来，这些按序排列的同类数据元素放在一起称为数组。

(数组中每一个元素有唯一的下标表示)

例如：

- 一个停车场，有1号停车位，2号停车位...n号停车位，每一个车位都有一个标号，每一个车位都可以停放一辆车，整个停车场相当于一个数组。

一维数组

一维数组可以想象成沿一条直线排列的数组，每个元素只需要一个下标就可以确定准确位置



一维数组

以下选项哪一个不可以看作是一个数组

- A、停车场内的20辆车
- B、30栋别墅
- C、超市货架上的20瓶农夫山泉
- D、书桌上的电脑、纸巾、鼠标、笔、笔记本、水杯

特别提示：数组可以存入任何相同类型的的数据

为什么要使用数组

存储一个数字，可以声明一个变量a来存储这个数字

```
int a;
```

存两个数字，可以声明两个变量a， b分别存下这两个数字

```
int a,b;
```

要存1000个数字呢？

```
int a1,a2,a3....a1000;
```

这样声明会非常麻烦

为什么要使用数组

这时候就可以使用数组，声明一个大小为1000的数组；

通过访问这个数组将这1000个数字存下

例如：int a[1000];

从这个例子可以看出，使用数组可以方便的处理大量数据

声明数组

在C++语言中，数组定义的格式是：

数据类型 数组名[数组大小] 注意：数组大小表示数组元素的个数，是一个 ≥ 0 的整数

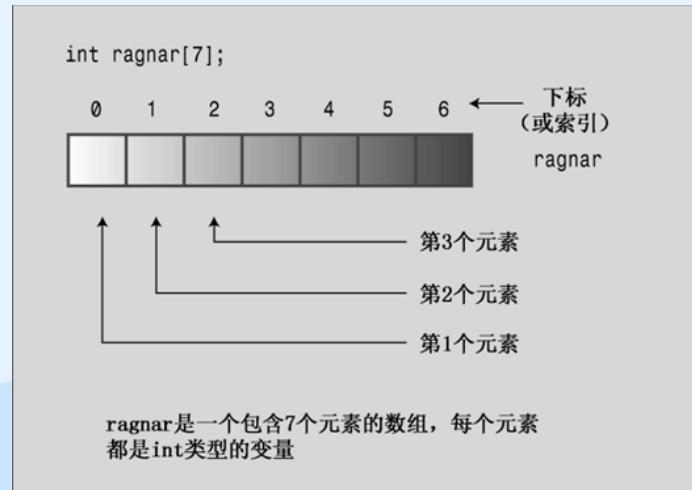
例如：int months[12];

在C++语言中，数组的编号从0开始。

声明数组

在C++语言中，数组的编号从0开始。

months[12]数组中第一个元素是months[0]，最后一个元素是months[11]



数组的初始化

1、用花括号括起，用逗号分隔的值列表（初始化列表）

例如：int a[3] = {1, 2, 3};

2、数组初始化时，初始的元素可以少于数组大小，编译器将其他元素默认为 0。

例如：int a[4] = {1, 2, 3};

则 a[0] = 1, a[1] = 2, a[2] = 3, a[3] = 0

3、如果不填写数组大小，自动计算元素个数并设置为数组大小

例如：int a[] = {5, 3, 1}; //a数组的大小为3

一维数组的使用

一维数组一般使用的形式：数组名[下标]
例如：a[3]

下标必须是一个整数，可以是常量、变量、表达式

● 例如：a[1]、a[i]、a[i+1] 都是合法

特别注意：

- 1、C++中数组下标是从0开始的，而不是1
- 2、数组a的第一个元素是a[0]
- 3、同一个数组，所有元素的数据类型都相同。
- 4、数组不能直接进行赋值，只有数组的元素可以赋值

练习

- 1、数组的初始化，以下那个是数组（ ）
A. char a[x+y]; B. int a[10];
- 2、找出错误的数组下标， $a[4]=\{1,2,3\}$ ，选项描述错误的是（ ）
A. a[1]=1; B. a[2]=3; C. A[4]=0;
- 3、数组的赋值练习，对应数据的位置；
数组 int a[10]={1,2,3,4,5,6,7,8,9,10}；那么a[7]=()
A. 7 B. 9 C. 8

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int a[5]; //定义一个元素个数为
               //5的数组
    cout << "Please input 5
numbers:"; //提示输入5个数
    for (int i = 0; i < 5; i++) //输入
        cin >> a[i];
    for (int i = 4; i >= 0; i--) //控制
        下标从4到0逆序输出
        cout << a[i] << " ";
    return 0;
}
```

一维数组使用举例

输入 5 个数然后将这 5 个数
倒序输出

...

输入 : 3 6 7 1 5

输出 : 5 1 7 6 3

...

2092 翻转数组

输入一个长度为n($1 \leq n \leq 100000$)数组，倒序输出他。
数组中的元素 a_i 满足($1 \leq a_i \leq 100000$)。

输入

- 第一行一个整数n，表示数字长度，
 $1 \leq n \leq 100000$
接下来n行，每行一个整数 a_i ，表示数组的内容， $1 \leq a_i \leq 100000$ 。

输出

- 输出第一行为数组长度n
接下来n行为倒序输出的结果。

3
4
5
6

3
6
5
4

2092 数组翻转 解题思路

逐个读入输入数据，并存储到数组 a 中，然后利用循环倒着遍历数组元素并输出。

2092 数组翻转

参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n; //读入n
    int a[n];
    for(int i = 0; i < n; i++)
        cin >> a[i]; //读入数组元素
    cout << n << endl;
    for(int i = n - 1; i >= 0;i--)
        cout << a[i] << endl;
    return 0;
}
```

1959 好事成双

中国人办喜事讲究好事成双，现在定义好事成双对：如果两个数字 a, b ，只要 $a=2b$ 或者 $b=2a$ 成立，那么就说这两个数字是好事成双对。现在给出一个数组，请计算一下里面有多少好事成双对。

样例解释：

1 4 3 2 9 7 18 22 0

这儿总共有3对：2和1，4和2，18和9。

输入

单组测试数据。

输入若干个正整数，这些整数不超过99，并且两两不同，最后一个数字是0，表示输入结束。

输出

输出一个整数表示答案。

样例输入1

1 4 3 2 9 7 18 22 0

样例输出1

3

1959 好事成双 解题思路

逐个读入数组元素，注意读到 0 就结束。

利用双重循环，逐个比较是否存在 $a_i = 2a_j$ ，如果存在，则计数 +1



1959 好事成双 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int a[100];
int main() {
    int n, ans = 0;
    for (int i = 0; i < 100; i++)
    {
        cin >> a[i];
        if(a[i] == 0) {
            n = i;
            break;
        }
    }
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            if(a[i] == a[j] * 2)
                ans++;
    cout << ans << endl;
    return 0;
}
```

3386 纸杯猜数

你和小华正在玩纸杯猜数游戏。

小华在桌面上扣放了一排 n 个纸杯，依次标号为 $1 \sim n$ 。每个纸杯内侧写有一个数字。

小华先将每个纸杯上的数字告诉小明，然后他快速的进行了如下 m 次交换操作：

选择两个纸杯 u 和 v ，交换它们的位置。（ u, v 相同表示本次不交换）

现在小华想让你回答： m 次操作后依次排列的纸杯上的数字分别是多少？

3386 纸杯猜数

输入

第一行输入两个数n,m，分别表示纸杯数、操作数（ $1 \leq n, m \leq 50000$ ）

第二行输入n个数，分别表示每个纸杯内侧的数字

之后m行，每行两个数u,v，表示将这两个纸杯交换。

输出

输出一行n个数，表示交换后每个纸杯内侧的数字依次是多少。以空格隔开。

8 3
1 4 7 20 5 3 11 6
3 7
2 6
5 3

1 3 5 20 11 4 7 6

3386 纸杯猜数 解题思路

简化一下题意，就是给出一个长度为 n 的数组，然后进行 m 次交换，最终输出交换后，每个位置的值。

- 因此按要求读入每个元素，然后循环 m 次，利用 3 变量交换法交换 2 个元素，最终输出结果即可
- 。



3386 纸杯猜数

参考答案

```
#include <bits/stdc++.h>
using namespace std;
int A[50010];
int main() {
    ios::sync_with_stdio(false);
    int n, m;
    cin >> n >> m;
    for(int i = 0; i < n; i++)
        cin >> A[i];
    for(int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        int t = A[a - 1]; A[a - 1] = A[b - 1]; A[b - 1] = t;
    }
    for(int i = 0; i < n; i++)
        cout << A[i] << " ";
    return 0;
}
```

数组的使用要点和 知识总结

- (1) 使用数组是为了方便处理大量数据
- (2) 同一个数组 , 所有元素的数据类型都相同。
- (3) 由于编译器不做下标的越界检查, 所以必须要注意下标的越界问题, 避免下标越界这样的错误发生。
- (4) 数组不能直接进行赋值操作 , 只有数组中的元素可以进行赋值操作 , “`int a[5], b[5];`”无法使用`b = a;`”进行赋值的。

易错知识点总结

1. 下标范围是从 0 到 (数组长度-1) ; 如果int a[2] , 只能用 a[0]和a[1]
2. 数组不能太大，局部变量与全局变量的存储空间

对于局部变量（函数中使用），数组大概10000个int--0.04M--（占用栈空间）

 - 对于全局变量，数组大概10000000个int --40M--（占用堆空间）
3. 计算空间的算法，1 int = 4Bytes, 1000000 int = 4MB.
4. 全局变量数组初始值是0，局部变量数组初始值不是0
如果需要赋初值，可以使用int a[5] = {};

数组和循环



课程安排

- 1、一维数组的读入与遍历
- 2、编程题目：2134逆序对
个数1000
- 3、编程题目：2129逆置换
- 4、编程题目：2419回文数

一维数组的读入

循环是操作数组的最好手段
，可以利用循环逐个读入数组的值

- 这段程序是先输入一个数n，
表示数组元素的数量

- 再输入n个数，程序会将这n
个数保存在数组a中

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    int a[10000]; //定义一个元素个数为5
    cin >> n;
    for (int i = 0; i < n; i++) //输入
        cin >> a[i];
    return 0;
}
```

遍历

使用循环，还可以方便的遍历数组，方便我们进行数据的处理。

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    int a[10000]; //定义一个元素个数为5
    cin >> n;
    for (int i = 0; i < n; i++) //输入
        cin >> a[i];
    for (int i = n - 1; i >= 0; i--) //输出
        if(a[i] % 2 == 0)
            cout << a[i] << endl;
    return 0;
}
```

以上程序将会读入n个数，然后倒序输出其中所有的偶数。

2134 逆序对个数1000

输入一个长度为n的数组，输出逆序对的个数，也就是说问有多少对 (i, j) 满足 $1 \leq i < j \leq n$ ，且 $a[i] > a[j]$ 。

$1 \leq n \leq 1000$

$1 \leq a[i] \leq 10^9$

输入

- 第一行一个整数n，表示数字长度
接下来n行，每行一个整数 $a[i]$ ，表示数组的内容。第一行一个

输出

输出一行一个数字，表示逆序对的个数。



2134逆序对个数1000（解题思路）

读入数组 a，利用双重循环逐个比较元素的数值，并做计数。注意，为了不重复统计，每个数只同后面的数字进行比较。



逆序对个数1000 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int n, a[1020], cnt;
int main() {
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> a[i];
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (a[i] > a[j])
                cnt++;
        }
    }
    cout << cnt << endl;
    return 0;
}
```

2129 逆置换

输入一个1到n的排列， $p[1], p[2], \dots, p[n]$ ，
即1到n都出现了1次的一个长度为n的数组p。
对于每个满足 $1 \leq i \leq n$ 的i，求下标j使得 $p[j] = i$ 。

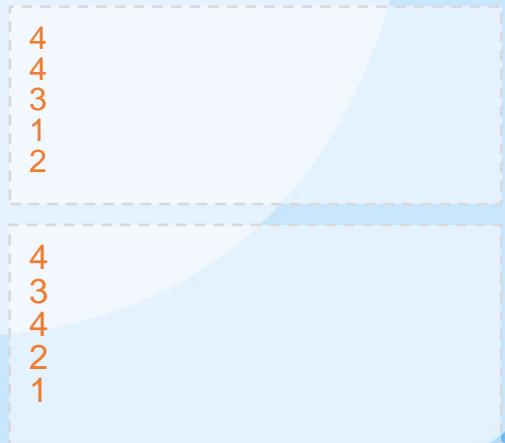
$1 \leq n \leq 100000$

输入

- 第一行一个整数n，表示排列长度
接下来n行，每行一个整数 $p[i]$ ，表示排列的内容
。

输出

- 第一行输出一个排列长度n，为了方便造数据。
输出共n行，其中第i行包含一个整数j，使得 $p[j]=i$
。



4
4
3
1
2

4
3
4
2
1

2129逆置换 解题思路

注意这道题里面的下标是从1开始计算的。

如果我们先读入数组，再逐个循环判断输出，需要双重循环。

按照数组的长度10w来计算，需要循环 100000×100000 次，这就太慢了，我们反过来，直接用数组存储每个值的下标。

ai中保存的是第几个被输入的。例如：

a3 = 5 表示3是第5个输入的数。

这样就可以直接输出最终结果了。



2129逆置换 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int a[200001];
int main() {
    int n, x;
    cin >> n;
    cout << n << endl;
    for (int i = 1; i <= n; i++) {
        cin >> x;
        a[x] = i;
    }
    for (int i = 1; i <= n; i++)
        cout << a[i] << endl;
    return 0;
}
```

2419 启蒙练习-回文数

数字回文的判定，如果一个数字从左往右看和从右往左看是一样的，那么我们称这个数字是回文数字，例如121,454454, 67876 , 5 , 22都是回文数字，而223 , 56 , 10就不是回文数字。现在读入一个数字X，请你写程序判断一下它是不是回文数字。

输入

输入一个正整数X

输出

如果X是回文数，输出"yes" ;
如果不是，输出"no"

334533

no

2419回文数（解题思路1）

将数字的每一位求出，并保存至数组a，设这个数字的长度为k，我们逐个比较：

$$a_1 = a_k$$

$$a_2 = a_{k-1}$$

.....

$$a_k = a_1$$

一旦出现不相等，则输出no，否则输出yes。



2419回文数 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int digs[15];
int main()
{
    int n, len = 0;
    cin >> n;
    while(n != 0) {
        digs[len] = n % 10;
        n /= 10;
        len++;
    }
    for(int i = 0; i * 2 <= len; i++) {
        if(digs[i] != digs[len - i - 1]) {
            cout << "no";
            return 0;
        }
    }
    cout << "yes";
    return 0;
}
```

2419回文数 解题思路2

如果学习了数组相关知识，可以很简单的解决这个问题，本题还有一个不用数组的巧妙解法。

解题思路是，求出当前数的镜像数，然后判断跟当前数是否相等，如果相等则表示当前数是回文数。

例如：

3124的镜像数为4213，而 3113的 镜像数为 3113。

利用 $/10$, $\%10$ 来求出x的每一位，同时将这一位记录到他的回文数中。

如何记录他的镜像数呢？

这边是不断 $/10$ ，并记录最后一位v，镜像数则是不断 $\times 10$ 并加上最后一位v。

2419回文数 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, t, rev = 0;
    cin >> n;
    t = n;
    while(t != 0)
    {
        int v = t % 10;
        rev = rev * 10 + v;
        t /= 10;
    }
    if(rev == n)
        cout << "yes";
    else
        cout << "no";
    return 0;
}
```

二维数组



课程安排

- 1、什么是二维数组
- 2、二维数组的声明及使用
- 3、二维数组的读入与遍历
- 4、数组与memset
- 5、编程题目：1504 二维数组转置
- 6、编程题目：3298 矩阵加法
- 7、编程题目：3214 完美方阵

二维数组是什么

二维数组本质上是以数组作为数组元素的数组，即“数组的数组”；
三维数组又称为矩阵。

例如：存储一个99乘法表；只使用一维数组实现起来是比较困难的，但使用二维数组就十分的方便。



$1 \times 1=1$				
$1 \times 2=2$	$2 \times 2=4$			
$1 \times 3=3$	$2 \times 3=6$	$3 \times 3=9$		
$1 \times 4=4$	$2 \times 4=8$	$3 \times 4=12$	$4 \times 4=16$	
$1 \times 5=5$	$2 \times 5=10$	$3 \times 5=15$	$4 \times 5=20$	$5 \times 5=25$
$1 \times 6=6$	$2 \times 6=12$	$3 \times 6=18$	$4 \times 6=24$	$5 \times 6=30$
$1 \times 7=7$	$2 \times 7=14$	$3 \times 7=21$	$4 \times 7=28$	$5 \times 7=35$
$1 \times 8=8$	$2 \times 8=16$	$3 \times 8=24$	$4 \times 8=32$	$5 \times 8=40$
$1 \times 9=9$	$2 \times 9=18$	$3 \times 9=27$	$4 \times 9=36$	$5 \times 9=45$
				$6 \times 9=54$
				$7 \times 9=63$
				$8 \times 9=72$
				$9 \times 9=81$

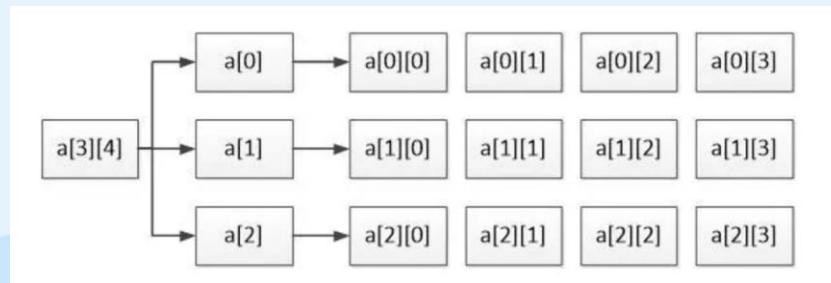
二维数组是什么

如何理解二维数组是“数组的数组”呢？

例如：

$a[3][4]$ ：可以理解为 $a[3][4]$ 这个二维数组有三个一维数组分别是 $a[0], a[1], a[2]$ ；每一个一维数组都是长度为4的一维数组；

可以抽象的理解为一个三行四列的矩阵



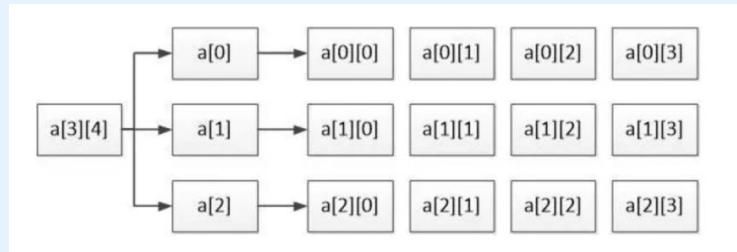
二维数组注意事项

注意： $a[0], a[1], a[2]$ 是数组名，不能当作下标变量使用；

二维数组下标在两个方向上变化。

二维数组是按行排列的：先存放 $a[0]$ 行，再存放 $a[1]$ 行，最后存放 $a[2]$ 行；每行中有四个元素也是依次存放。

由于数组 a 为int类型，该类型占4个字节的内存空间，所以每个元素均占有4个字节。



二维数组的声明

二维数组的声明：

数据类型 数组名[正整数][正整数];

注意：一维数组的规则一样，数组大小必须是正整数。

例1：

```
int a[3][4];
```



二维数组的初始化

1、按行给二维数组赋初值（括号套括号形式）

```
int a[2][3] = {{1,2,3},{4,5,6}};
```

[2]表示：两行；[3]表示：每行有3个元素

			第一个元素	第一个元素	第一个元素
a[2][3]	第一行	a[0]	a[0][0]	a[0][1]	a[0][2]
			=1	=2	=3
	第二行	a[1]	a[1][0]	a[1][1]	a[1][2]
			=4	=5	=6

二维数组的初始化

2、将所有的数组元素按行顺序写在一个大括号内

```
int a[2][3] = {1,2,3,4,5,6};
```

二维数组共有两行，每行有三个元素；

第一行的元素依次为1, 2, 3；第二行元素依次为
4, 5, 6



二维数组的初始化

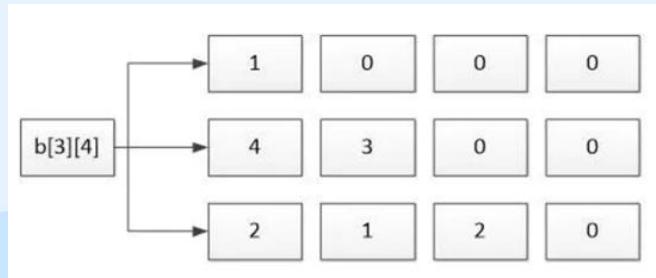
3、对部分数组元素赋初值

```
int b[3][4] = {{1},{4,3},{2,1,2}};
```

只对数组中的部分元素进行了赋值操作，没有赋值的自动赋值为0

。

具体参看下图：



二维数组的初始化

特殊情况：

二维数组的第一个下标可省略

例如：

- ```
int a[][3] = {1,2,3,4,5,6}; → int a[][3] =
{1,2,3,4,5,6};
```



# 二维数组的引用

二维数组中，每一个元素则由唯一的两个下标确定。

二维数组的元素也称为双下标变量，形式为：

数组名[下标1][下标2]

例如： a[2][3]



## 单选题

下面二维数组赋初值正确的是( )

- A int a[2][2]={‘123’ , ’345’};
- B int a[2][3] = {{1,2,3},{4,5,6}};

## 二维数组的读入

读入一个 $100 \times 100$ 的二维数组

```
#include <bits/stdc++.h>
using namespace std;
int main() {
 int a[100][100];
 for (int i = 0; i < 100; i++)
 {
 for(int j = 0; j < 100; j++)
 {
 cin >> a[i][j];
 }
 }
 return 0;
}
```

## 二维数组的遍历

读入一个 $100 \times 100$ 的二维数组，然后输出这个数组

```
#include <bits/stdc++.h>
using namespace std;
#include <bits/stdc++.h>
using namespace std;
int main() {
 int a[100][100];
 for (int i = 0; i < 100; i++) {
 for(int j = 0; j < 100; j++) {
 cin >> a[i][j];
 }
 }
 for (int i = 0; i < 100; i++) {
 for(int j = 0; j < 100; j++) {
 cout << a[i][j] << " ";
 }
 cout << endl;
 }
 return 0;
}
```

## 二维数组的初始化赋值

```
#include <bits/stdc++.h>
using namespace std;
int main ()
{
 /* 一个带有 5 行 2 列的数组 */
 int a[5][2] = { {0,0}, {1,2}, {2,4},
{3,6},{4,8} };
 /* 输出数组中每个元素的值 */
 for (int i = 0; i < 5; i++)
 {
 for (int j = 0; j < 2; j++)
 cout << "a[" << i << "][" << j << "]"
= " " << a[i][j] << endl;
 }
 return 0;
}
```

## 二维数组的初始化赋值

```
#include <bits/stdc++.h>
using namespace std;
int main ()
{
 /* 一个带有 5 行 2 列的数组 */
 int a[5][2] = { {0,0}, {1,2}, {2,4},
 {3,6},{4,8}};

 /* 输出数组中每个元素的值 */
 for (int i = 0; i < 5; i++)
 {
 for (int j = 0; j < 2; j++)
 cout << "a[" << i << "][" << j
<< "] = " << a[i][j] << endl;
 }
 return 0;
}
```

输出结果为：

a[0][0] = 0  
a[0][1] = 0  
a[1][0] = 1  
a[1][1] = 2  
a[2][0] = 2  
a[2][1] = 4  
a[3][0] = 3  
a[3][1] = 6  
a[4][0] = 4  
a[4][1] = 8

## 数组与memset

memset是指使用很大数组且都赋值为0时，使用memset方便简洁不易出错；

格式为：

memset(数组名, 赋值, 数组大小);

例如：

```
#include <bits/stdc++.h>
using namespace std;
int data1[1000];
double dataD[1000];
int data2[100][100];
int main()
{
 memset(data1, 1, sizeof(data1)); // 将data1赋初始值为1（以字节为单位）
 memset(dataD, 0, sizeof(dataD)); // 将double数组赋初始值为0
 memset(data2, 0, sizeof(data2)); // 将二维数组赋初始值为0
}
```

# 1504 二维数组转置

输入一个n行m列的数组，输出他的转置，具体来说：

输出的第i行第j个数字，应是输入的第j行第i个数字。

$1 \leq n \leq 100$

$1 \leq m \leq 100$

$1 \leq a[i][j] \leq 1000$

## 输入

- 第一行两个整数n, m表示数组的行数和列数  
接下来n行，每行m个整数表示数组内容。

## 输出

第一行先输出m, n。

接下来输出转置的结果，共m行n列。

其中第i行第j个数字，应是输入的第j行第i个数字。

|   |   |
|---|---|
| 3 | 2 |
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |

|   |   |   |
|---|---|---|
| 2 | 3 |   |
| 1 | 3 | 5 |
| 2 | 4 | 6 |

# 1504 二维数组转置（解题思路）

先利用双重循环读入二维数g。

转置本质是将 $g[i][j]$ 放到 $g[j][i]$ 的位置，可以开一个新的数组，记录转置的结果，然后利用双重循环输出这个数组即可。



# 二维数组转置

## 参考答案

```
#include <bits/stdc++.h>
using namespace std;
const int N = 110;
int g[N][N];
int main(){
 int n,m;
 cin >> n >> m;
 for(int i = 0;i < n; i++){
 for(int j = 0;j < m; j++){
 cin >> g[i][j];
 }
 }
 cout << m << ' ' << n << endl;
 for(int i = 0;i < m; i++){
 for(int j = 0;j < n; j++){
 cout << g[j][i]<< ' ';
 }
 cout << endl;
 }
}
```

# 3298 矩阵加法

输入两个n行m列的矩阵A和B，输出它们的和A+B。

# 3298 矩阵加法

## 输入

第一行包含两个整数n和m，表示矩阵的行数和列数。 $1 \leq n \leq 100$ ， $1 \leq m \leq 100$ 。

接下来n行，每行m个整数，表示矩阵A的元素。

接下来n行，每行m个整数，表示矩阵B的元素。

相邻两个整数之间用单个空格隔开，每个元素均在1~1000之间。

## 输出

n行，每行m个整数，表示矩阵加法的结果。相邻两个整数之间用单个空格隔开。

|       |
|-------|
| 3 3   |
| 1 2 3 |
| 1 2 3 |
| 1 2 3 |
| 1 2 3 |
| 4 5 6 |
| 7 8 9 |

|         |
|---------|
| 2 4 6   |
| 5 7 9   |
| 8 10 12 |

## 3298 矩阵加法（解题思路）

矩阵的加法运算很简单，就是将对应位置的值相加即可，因此按照要求定义 2 个二维数组，并读入数值。

然后利用双重循环，输出元素的和即可。

# 3298 矩阵加法 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int C1[105][105], C2[105][105];
int main() {
 int n, m;
 cin >> n >> m;
 for(int i = 0; i < n; i++)
 for(int j = 0; j < m; j++)
 cin >> C1[i][j];
 for(int i = 0; i < n; i++)
 for(int j = 0; j < m; j++)
 cin >> C2[i][j];
 for(int i = 0; i < n; i++) {
 for(int j = 0; j < m; j++)
 cout << C1[i][j] + C2[i][j] << " ";
 cout << endl;
 }
 return 0;
}
```

# 3214 完美方阵

小明很喜欢矩阵，在他的心目中这样的矩阵能够被称为完美方阵：这个方阵的n行与n列当中，每行的数字之和必须相同，每列的数字之和也必须相同。

现在你找来一些方阵，你想知道小明是否愿称之为完美方阵。

## 输入

- 第一行输入一个数n，表示一个方阵的行列数；之后n行，每行n个数，表示方阵中的每个元素  $a[i][j]$ ，以空格隔开；

## 输出

若这个方阵是完美方阵，输出“YES”，否则输出“NO”。

|       |
|-------|
| 3     |
| 3 4 8 |
| 7 2 6 |
| 5 9 1 |

YES

## 3214 完美方阵（解题思路）

根据题目要求，我们先要知道这个和究竟是多少，因此先利用一重循环计算这个和 sum

然后利用双重循环，对每一行每一列求和，如果某行或者某列所有行列的和不等于 sum。, 则输出 No

如果处理到最后，仍然都满足条件，则输出 Yes

# 3214 完美方阵 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int a[51][51];
int main() {
 int n, sum = 0;
 cin >> n;
 for (int i = 0; i < n; i++)
 for (int j = 0; j < n; j++)
 cin >> a[i][j];
 for (int i = 0; i < n; i++)
 sum += a[0][i];
 for (int i = 0; i < n; i++) {
 int cSum = 0;
 for (int j = 0; j < n; j++)
 cSum += a[i][j];
 if (cSum != sum) {
 cout << "No" << endl;
 return 0;
 }
 }
 for (int i = 0; i < n; i++) {
 int cSum = 0;
 for (int j = 0; j < n; j++)
 cSum += a[j][i];
 if (cSum != sum) {
 cout << "No" << endl;
 return 0;
 }
 }
 cout << "Yes" << endl;
 return 0;
}
```

# 前缀和



## 课程安排

- 1、编程题目：2094前缀和
- 2、前缀和应用
- 3、编程题目：1545区间和
- 4、编程题目：1505子矩阵求和

# 什么是前缀和

前缀和中的第 $i$ 项，表示原数组中的前 $i$ 项的和。



# 2094 前缀和

输入一个长度为n( $1 \leq n \leq 100000$ )数组 $a_i(0 \leq a_i \leq 1000)$ ，输出他的前缀和。

前缀和中的第*i*项，表示原数组中的前*i*项的和。

## 输入

- 第一行一个整数n，表示数字长度  
接下来n行，每行一个整数 $a_i$ ，表示数组的内容  
。

## 输出

- 输出第一行为数组长度n  
接下来n行为前缀和的结果。

3  
1  
2  
3

3  
1  
3  
6

# 2094前缀和 解题思路

利用循环读入数据，同时求和即可。

求前缀和甚至可以不使用数组，仅仅依靠循环就可以实现。



# 2094前缀和 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main() {
 int n, v, sum = 0;
 cin >> n;
 cout << n << endl;
 for (int i = 0; i < n; i++) {
 cin >> v;
 sum += v;
 cout << sum << endl;
 }
 return 0;
}
```

# 前缀和应用

给定一个数组  $A[1..n]$  , 前缀和数组  $\text{PrefixSum}[1..n]$  定义为 :

\$\$

$$\text{PrefixSum}[i] = A[0]+A[1]+\dots+A[i-1]$$

\$\$

例如 :  $A[5,6,7,8] \rightarrow \text{PrefixSum}[5,11,18,26]$

# 前缀和应用

那么当我们预处理出前缀和后，就可以快速求得任意的区间和了，对于一个区间 $[l, r]$ ：

$$A[l] + A[l+1] + A[l+2] + \dots + A[r] = \text{PrefixSum}[r] - \text{PrefixSum}[l-1]$$

最简单的一道题就是给定n个数和m次询问，每次询问一段区间的和。

不用每次都循环整个区间进行统计，做好预处理后，可以快速的计算区间和。

# 前缀和应用

```
for (int i = 1; i <= n; i++)
 PrefixSum[i] = PrefixSum[i - 1] +
A[i];
while (m--) {
 int L, R;
 cin >> L >> R;
 cout << PrefixSum[R] - PrefixSum[L
- 1] << endl;
}
```

# 1545 区间和

输入一个长度为n( $1 \leq n \leq 1000$ )的数组a，元素为 $a[1] \dots a[n]$ ，之后进行m次询问，每次询问给出两个值(l,r) ( $r \geq l$ )，求数组： $a[l] + a[l+1] + \dots + a[r]$ 的值。

# 1545 区间和

## 输入

第一行2个数，n和m，中间用空格分隔（ $1 \leq n, m \leq 1000$ ）。

之后 $n+m$ 行，

第1至n行：每行一个数字 $a[i]$ （ $0 \leq a[i] \leq 1000$ ）

- 第 $n+1$ 至 $n+m$ 行：每行2个数字 $l, r$ ，中间用空格分隔（ $0 < l \leq r \leq n$ ）

## 输出

输出共m行，每行一个数，对应 $a[l] + a[l+1] + \dots + a[r]$ 的值。

|     |   |
|-----|---|
| 3 3 |   |
| 1   |   |
| 3   |   |
| 5   |   |
| 1 2 |   |
| 1 3 |   |
| 2 3 |   |
|     | 4 |
|     | 9 |
|     | 8 |

# 1545区间和 解题思路

利用预处理，保存数组的前缀和，之后对于每一个询问，可以直接计算出区间和。



# 1545区间和 参考答案

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e3 + 10;
int a[N];
int main() {
 int n, m, l, r;
 cin >> n >> m;
 for (int i = 1; i <= n; ++i)
 cin >> a[i];
 for (int i = 1; i <= n; ++i)
 a[i] += a[i - 1];
 for (int i = 1; i <= m; ++i) {
 cin >> l >> r;
 cout << a[r] - a[l - 1] << endl;
 }
 return 0;
}
```

# 1505子矩阵求和

题目描述：

给出一个 $m * n$ 的矩阵a，矩阵元素 $a[i,j]$ 小于1000，进行q次查询，每次查询给出子矩阵的4个边界（上下左右），求该子矩阵所有元素之和。样例中第一个查询：1 3 1 2 表示从第1行到第3行，从第1列到第2列，对应的子矩阵是：

1 2

5 6

9 10

求和等于33

输出共q行，对应q个询问的求和结果。

# 1505 子矩阵求和

## 输入

第一行2个整数n, m , 中间用空格分割，分别对应数组的行数n、列数m( $1 \leq n, m \leq 100$ )

接下来n行，每行m个整数表示矩阵的内容 $a[i,j]$ 。 $(0 \leq a[i,j] \leq 1000)$

接下来1行，一个数q，对应查询的数量。 $(1 \leq q \leq 1000)$

接下来q行，每行4个整数，对应矩阵的上下左右边界u,d,l,r。 $(1 \leq u \leq d \leq n, 1 \leq l \leq r \leq m)$

## 输出

输出共q行，对应q个询问的求和结果。

|            |
|------------|
| 3 4        |
| 1 2 3 4    |
| 5 6 7 8    |
| 9 10 11 12 |
| 3          |
| 1 3 1 2    |
| 1 2 1 3    |
| 1 3 1 3    |

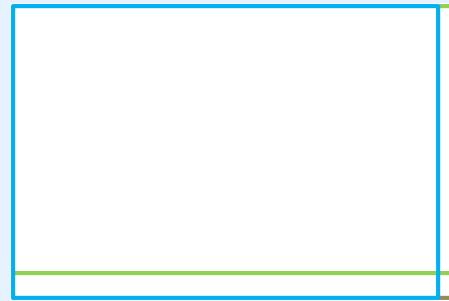
|    |
|----|
| 33 |
| 24 |
| 54 |

# 1505子矩阵求和 解题思路

看一下这个图：

那么最大的矩形前缀和就等于  
蓝的矩阵加上绿的矩阵，再减  
去重叠面积，最后加上小方块  
，即

$$\text{sum}[i][j] = \text{sum}[i][j - 1] + \text{sum}[i - 1][j] - \text{sum}[i - 1][j - 1] + a[i][j]$$



这样我们可以快速求出所有以(0,0)为左上角，(i,j)为右下角的矩阵前缀和。

```
for(int i = 1; i <= n; i++)
```

```
 for(int j = 1; j <= n; j++)
```

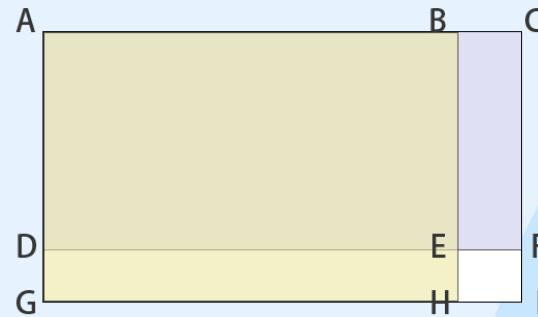
$$\text{sum}[i][j] = \text{sum}[i - 1][j] + \text{sum}[i][j - 1] - \text{sum}[i - 1][j - 1] + a[i][j];$$

# 1505子矩阵求和 解题思路

再来看一下这个图：

仍然利用上面给出的容斥的方法：矩阵(E,H,I,F)的和等于：

$$(A,G,I,C) - (A,G,H,B) - (A,D,F,C) + (A,D,E,B)$$



转换为题目对应的矩阵，就是： $\text{sum}[d][r] - \text{sum}[d][l-1] - \text{sum}[u-1][r] + \text{sum}[u-1][l-1]$ ;

因此对于任意一个子矩阵，我们都可以快速的对其求和。

# 1505子矩阵求和 参考答案

```
#include <bits/stdc++.h> //头文件
using namespace std; //命名空间
int main() //主函数声明
{
 int n, m, v, sum[105][105];
 cin >> n >> m;
 for(int i = 1; i <= n; i++){
 for(int j = 1; j <= m; j++){
 cin >> v;
 sum[i][j] = sum[i - 1][j] +
sum[i][j - 1] - sum[i - 1][j - 1] + v;
 }
 }
}
```

```
int q;
cin >> q;
int u, d, l, r;
for(int i = 0; i < q; i++){
 long long total = 0;
 cin >> u >> d >> l >> r;
 total = sum[d][r] - sum[d][l-1]
- sum[u-1][r] + sum[u-1][l-1];
 cout << total << endl;
}
return 0;
```

# 二进制



# 课程安排

- |                           |                        |
|---------------------------|------------------------|
| 1、进制认知，进制的定义              | 6、编程题目：2112转二进制        |
| 2、二进制的原码，反码               | 7、编程题目：2121转二进制2       |
| 3、正数十进制转二进制，<br>正数二进制转十进制 | 8、编程题目：3324二进制<br>的各种码 |
| 4、二进制加减法                  | 9、头脑风暴：基因药水            |
| 5、二进制中的小数                 |                        |

# 进制认知

同学们咱们数学启蒙中最先学习的数字是不是：0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11..... ?

大家观察一下，我们数到几从个位（1位）升级成十位（2位）的？

.....从9到10的时候，我们用到数位从个位（1位）升级到了十位（2位）对不对？

里我们用到了每次数到10就要进一位对不对？  
这就是十进制（逢十进一）



# 进制认知

我们在思考一下生活中的时间计数：都用到了哪些进位方法？

1年=12个月----- (逢\_\_\_\_\_进1)

1天=24小时----- (逢\_\_\_\_\_进1)

1小时=60分钟----- (逢\_\_\_\_\_进1)

1分钟=60秒 ----- (逢\_\_\_\_\_进1)

那么，这些都用了哪些进制？

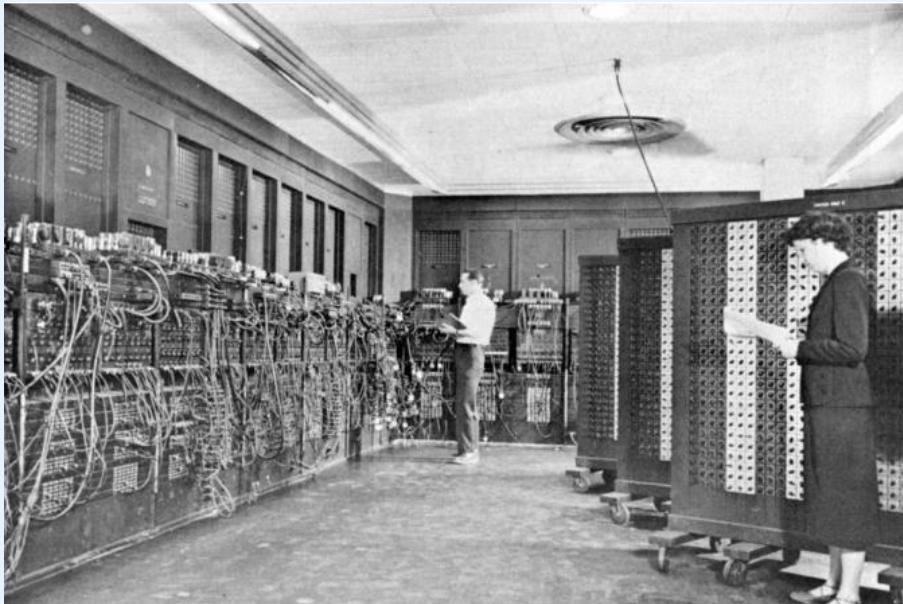
- A.十进制
- B.二进制
- C.12进制
- D.60进制
- E.24进制

# 进制认知

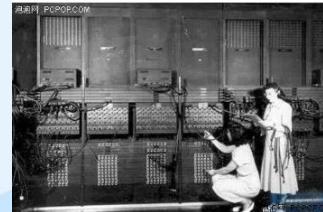
大家见过世界上的第一台计算机吗？  
第一台计算机是怎么操作和计算的呐？



# 第一台计算机



|     |           |
|-----|-----------|
| 关键词 | 计算机诞生     |
| 目的  | 弹道计算      |
| 日期  | 1946-2-14 |
| 类型  | 电子管       |
| 发名人 | 美国人莫克利    |
| 名字  | ENIAC     |



# 计算机语言与二进制

机器语言：  
直接指令（0、1打孔带）

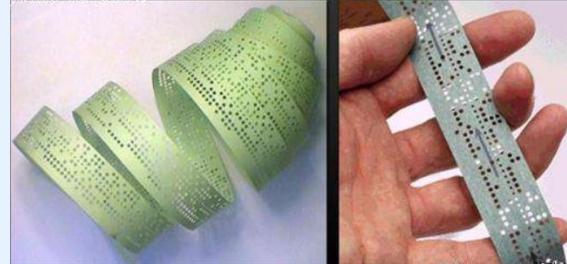


表 7-1 机器语言程序示例

| 序号 | 指 令       |           |                               |
|----|-----------|-----------|-------------------------------|
| 1  | 0000 0000 | 0000 0100 | 0000 0000 0000 0000           |
| 2  | 0101 1110 | 0000 1100 | 1100 0010 0000 0000 0000 0010 |
| 3  |           | 1110 1111 | 0001 0110 0000 0000 0000 1011 |
| 4  |           | 1110 1111 | 1001 1110 0000 0000 0000 1011 |
| 5  | 1111 1000 | 1010 1101 | 1101 1111 0000 0000 0001 0010 |
| 6  |           | 0110 0010 | 1101 1111 0000 0000 0001 0101 |
| 7  | 1110 1111 | 0000 0010 | 1111 1011 0000 0000 0001 0111 |
| 8  | 1111 0100 | 1010 1101 | 1101 1111 0000 0000 0001 1110 |
| 9  | 0000 0011 | 1010 0010 | 1101 1111 0000 0000 0010 0001 |
| 10 | 1110 1111 | 0000 0010 | 1111 1011 0000 0000 0010 0100 |
| 11 | 0111 1110 | 1111 0100 | 1010 1101                     |
| 12 | 1111 1000 | 1010 1110 | 1100 0101 0000 0000 0010 1011 |
| 13 | 0000 0110 | 1010 0010 | 1111 1011 0000 0000 0011 0100 |
| 14 | 1110 1111 | 0000 0010 | 1111 1011 0000 0000 0011 0100 |
| 15 |           |           | 0000 0100 0000 0000 0011 1101 |
| 16 |           |           | 0000 0100 0000 0000 0011 1101 |

# 进制的定义

| 进制名称     | 二进制   | 八进制             | 十六进制                            |
|----------|-------|-----------------|---------------------------------|
| 规则       | 逢二进一  | 逢八进一            | 逢十六进一                           |
| 表示方式     | 0 , 1 | 0,1,2,3,4,5,6,7 | 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F |
| 编程语言表示方式 |       | 数字0开头           | 用0X开头                           |
| 举例 : 13  | 1101  | 15              | D                               |

## 二进制中的原码、反码

对于正负数（有符号数）：

二进制的最高位是符号位：0表示正数，1表示负数

正数的原码、反码都一样

负数的反码 = 它的原码符号位不变，其他位取反（ $0 \rightarrow 1$ ； $1 \rightarrow 0$ ）

0的反码还是0

# 正数十进制转二进制

转换规则：除二取余，倒序排序，高位补0（短除法）

例：将25转换为二进制数

解：

$$25 \div 2 = 12 \text{ 余数 } 1$$

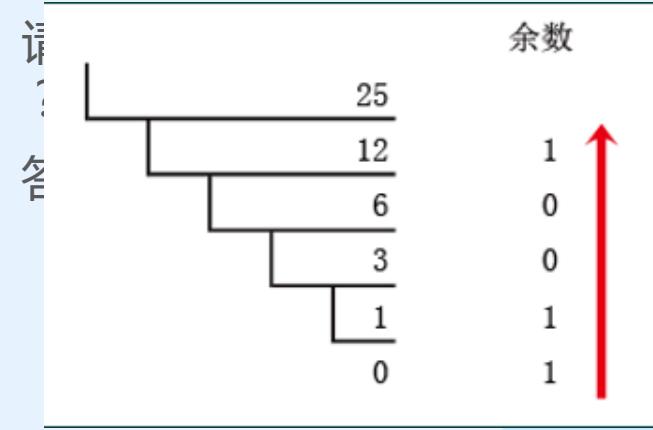
$$12 \div 2 = 6 \text{ 余数 } 0$$

$$6 \div 2 = 3 \text{ 余数 } 0$$

$$3 \div 2 = 1 \text{ 余数 } 1$$

$$1 \div 2 = 0 \text{ 余数 } 1$$

所以  $25 = (11001)_2$



# 正数二进制转十进制

二进制数1000110转成十进制数可以看作这样：

数字中共有三个1 即第一位一个，第二位一个，第六位一个，然后对应十进制数即：  
 $2^1 + 2^2 + 2^6$ ，

则

$$\begin{aligned}1000110 &= 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 \\&+ 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\&= 64 + 0 + 0 + 0 + 4 + 2 + 0 \\&= 70\end{aligned}$$

请同学们将1110转换为十进制数？

答案：

# 二进制加减法

二进制的加法计算原则为：

- 1、把二进制转为十进制
- 2、进行加法计算
- 3、十进制结果转为二进制

例如：计算 $110+111$

答：

$$110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2 + 0 = 6$$

$$111 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 4 + 2 + 1 = 7$$

13转为二进制为： $1101$

$$110 + 111 = 1101$$

二进制的减法计算原则为：

- 1、把二进制转为十进制
- 2、进行减法计算
- 3、十进制结果转为二进制

例如：计算 $1101 - 111$

答：

$$\begin{aligned} 1101 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 \\ &+ 4 + 0 + 1 = 13 \end{aligned}$$

$$111 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 4 + 2 + 1 = 7$$

6转为二进制为： $110$

$$1101 - 111 = 1101$$

# 二进制中的小数

小数进制转换原则：只计算小数点后面的位

$$2^{-1} = 0.5$$

$$2^{-2} = 0.25$$

$$2^{-3} = 0.125$$

$$2^{-4} = 0.0625$$

$$2^{-5} = 0.03125$$

| 二进制    | 十进制                                                                        |
|--------|----------------------------------------------------------------------------|
| 0.1    | $1 \times 2^{-1} = 0.5$                                                    |
| 0.0101 | $0.5 \times 0 + 0.25 \times 1 + 0.125 \times 0 + 0.0625 \times 1 = 0.3125$ |

# 2112 转二进制

请你把一个整数n转化为二进制并从低位到高位输出。

## 输入

- 一行一个整数n，保证 $1 \leq n \leq 10^9$ 。

## 输出

从低位到高位输出一个二进制数，表示n的二进制形式，每位之间不需要空格

11

1101

# 2112 转二进制 解题思路

由于是从低位到高位输出，因此直接利用短除法即可。



# 2112 转二进制 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main() {
 int n;
 cin >> n;
 while(n != 0) {
 cout << n % 2;
 n /= 2;
 }
 return 0;
}
```

# 2121 转二进制2

请你把一个整数n转化为二进制并从高位到低位输出。

## 输入

一行一个整数n，保证 $1 \leq n \leq 10^9$ 。

## 输出

从高位到低位输出一个二进制数，表示n的二进制形式，每位之间不需要空格。

11

1011

# 2121 转二进制2 解题思路

利用短除法得到的二进制表示，是从低位到高位的，而题目要求从高位到低位输出，所以先利用短除法把二进制的每一位保存到数组中，再倒序输出出来即可。



## 2121 转二进制2 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int nums[32];
int main() {
 int n, max = 0;
 cin >> n;
 while(n > 0)
 {
 nums[max] = n % 2;
 n /= 2;
 max++;
 }
 for(int i = max - 1; i >= 0; i--)
 cout << nums[i];
 cout << endl;
 return 0;
}
```

## 3324 二进制的各种码

原码：用第一位表示符号，其余位表示值。正数的符号位为0，负数为1。例如1的8位原码是[00000001]，-1的8位原码是[10000001]。

反码：正数的反码是其本身，负数的反码是符号位保持不变，其余位取反。例如-1的8位反码是[11111110]。

补码：正数的补码是其本身，负数的补码是在其反码的基础上+1，例如-1的补码是[11111111]。

输入一个整型(int)的整数，输出如下内容：

1. 它的32位二进制原码。
2. 它的32位二进制反码。
3. 它的32位二进制补码。



# 3324 二进制的各种码 解题思路

根据原码，反码，补码的定义，输出每一位，这里有个简单的处理输入的方法。

```
cout << bitset<32>(n) << endl;
```

可以按位输出每一位的值。

```
cout << bitset<31>(n) << endl;
```

则只输出31位二进制。



## 二进制的各种码 参考答案

```
#include <bits/stdc++.h>
using namespace std;
int main() {
 int n;
 cin >> n;
 if(n >= 0) {
 cout << bitset<32>(n) << endl;
 cout << bitset<32>(n) << endl;
 cout << bitset<32>(n) << endl;
 }
 else {
 cout << 1 << bitset<31>(-n) << endl;
 cout << bitset<32>(n - 1) << endl;
 cout << bitset<32>(n) << endl;
 }
 return 0;
}
```

# 基因药水

这次头脑风暴，我们带大家完成一个很有名的验证学问题，这个问题中所涉及的思想对我们算法竞赛有很强的指导性意义。

题目描述：

现在实验室中有60瓶药水，这60瓶药水当中，有一瓶是特殊的基因药水。一旦小白鼠喝了这种基因药水，就会在第二天变成黑色，而其他的 59瓶药水都是普通的纯净水，小白鼠喝了之后不会有任何变化。

现在你只有一天时间了，最少要用多少只小白鼠才能在第二天通过观察小白鼠的变色结果，判断出哪一瓶药水是基因药水。

# 基因药水

对于这道题，不难思考：如果有60只小白鼠，是保证能完成任务的，因为这样的话，我们可以把60瓶药水分别喂给这60只小白鼠，想观察哪一只小白鼠变色了就能得出试验结果。

但是，60显然定不是我们想要的最佳答案，我们很想让尽量少的小白鼠能给我们提供足够多的信息。

这个时候可以引入二进制的思想，因为这60瓶药水分别编号为1到60，而每一个编号，如果把它拆解成二进制，都能用一个长度不超过6的01字符串来表示。

比如1表示成二进制是000001，21表示成二进制是010101，60表示成二进制是111100。

# 基因药水

通过60个不同的字符串，对编号1到60做了区分。

这样有什么用呢，这样一来。

假如：用六只小白鼠，分别将这60瓶药他们编号所对应的二进制字符串为规则，喂给对应的小白鼠，第二天观察小白鼠颜色的变化，基因药水对应的那一瓶的编号二进制位的小白鼠一定都是变成黑色的，而其他的药水不会对小白鼠产生变化。

这样通过观察小白鼠颜色的分布，就能确定基因药水的编号了

。

# 总结

我们一起学习了进制认知，进制的定义；二进制的原码，反码；正数十进制转二进制，正数二进制转十进制；二进制加减法；二进制中的小数。在NOIP的复赛编程中会经常考察。

