



# C++编程



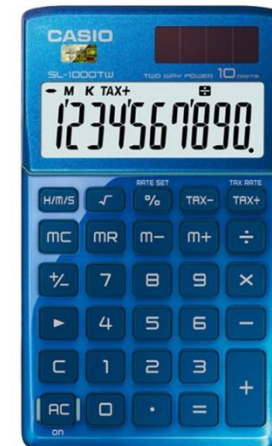
# 目录

- 函数
- 递归
- 不定长输入
- 无穷大
- 排序和去重
- 埃氏筛
- 线性筛
- 二分查找
- 在线和离线
- 二维数组
- 字符数组
- 字符串
- 高精度运算
- 结构体
- 文件操作
- 快速读入
- 位运算

# 高精度乘法

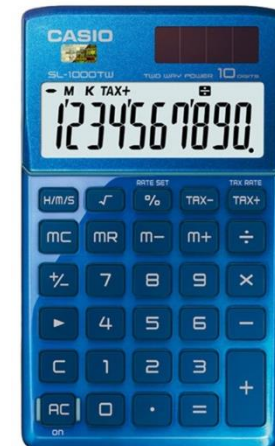
- $A \times B$  problem

$$1 \leq A, B \leq 10^{10000}$$



Sample input	Sample output
2147483647 2019	4335769483293

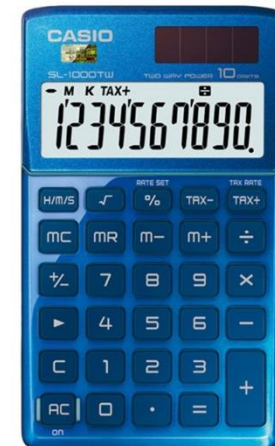
# 高精度乘法



- 高精度乘法也是模拟一遍竖式乘法

$$\begin{array}{r} \phantom{00}456 \\ \times 789 \\ \hline 359784 \end{array}$$

# 高精度乘法



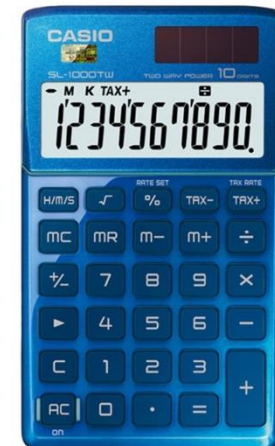
- 但是竖式乘法比竖式加法复杂多了

1. 先对位乘 $456 \times 9$

这里不着急进位，因为不如最后统一进位

$$\begin{array}{r} \begin{array}{|c|c|c|} \hline 4 & 5 & 6 \\ \hline \end{array} \\ \times \quad \begin{array}{|c|c|c|} \hline & & 9 \\ \hline \end{array} \\ \hline \begin{array}{|c|c|c|c|c|c|} \hline & & & 3 & 6 & 4 & 5 & 5 & 4 \\ \hline \end{array} \end{array}$$

# 高精度乘法



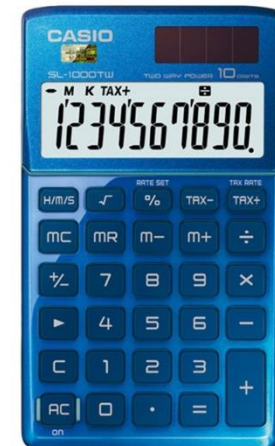
- 但是竖式乘法比竖式加法复杂多了

2. 再对位乘 $456 \times 8$

这里不着急进位，因为不如最后统一进位

			4	5	6
				8	
<hr/>					
			36	45	54
		32	40	48	

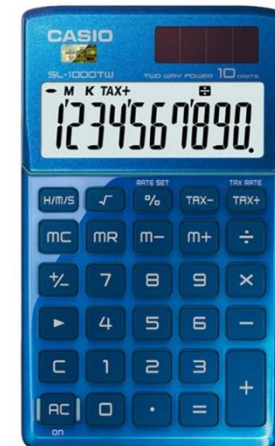
# 高精度乘法



- 但是竖式乘法比竖式加法复杂多了
3. 最后对位乘 $456 \times 7$

			4	5	6
			7		
X					
<hr/>					
			36	45	54
		32	40	48	
	28	35	42		

# 高精度乘法



- 但是竖式乘法比竖式加法复杂多了

3. 最后对位乘 $456 \times 7$

但是怎么把积放在对应的位置呢？

回忆一下多重循环的执行流程

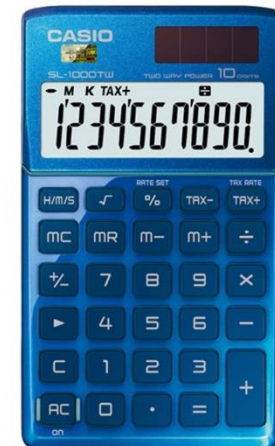
```
for (i: 0 ~ a最后一位)
    for (j: 0 ~ b最后一位)
        c[i+j] += a[i] * b[j]
```

	4	5	6
X	7		

			364554		
		324048			
	283542				



# 高精度乘法



- 但是竖式乘法比竖式加法复杂多了

4. 把和全部加起来

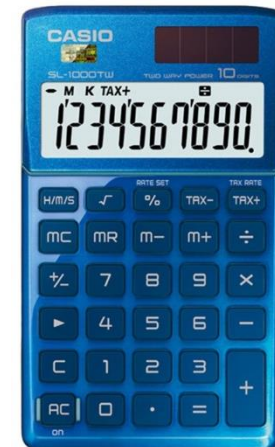
这里再用高精度加法同样的做法统一进位

```
for (i: 0 ~ a最后一位)
    for (j: 0 ~ b最后一位)
        c[i+j] += a[i] * b[j]
```

$$\begin{array}{r} 456 \\ \times 789 \\ \hline \end{array}$$

	286716454				
		324048			
	283542				

# 高精度乘法



- 但是竖式乘法比竖式加法复杂多了

4. 把和全部加起来

这里再用高进度加法同样的做法统一进位

$$\begin{array}{r} \begin{array}{|c|c|c|} \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array} \\ \times \\ \hline \begin{array}{|c|c|c|c|c|c|} \hline 3 & 5 & 9 & 7 & 8 & 4 \\ \hline \end{array} \end{array}$$

# 参考代码

```
#include<bits/stdc++.h>
using namespace std;
string multi(string A, string B)
{
    string ans;
    int i, j, a[10010], b[10010], c[20010];
    int LA = A.size(), LB = B.size();
    for (i = LA - 1; i >= 0; i --) a[LA-1-i] = A[i] - '0';
    for (i = LB - 1; i >= 0; i --) b[LB-1-i] = B[i] - '0';
    for (i = 0; i < LA; i ++)
        for (j = 0; j < LB; j ++)
            c[i+j] += a[i] * b[j];    //对位乘
    for (i = 0; i < LA + LB - 1; i ++) //处理 c[] 中的进位
        c[i+1] += c[i] / 10, c[i] = c[i] % 10;
    if (c[i]) i ++;    //判断最高位是否进位
    for (i = i - 1; i >= 0; i --) ans += c[i] + '0';
    return ans;
}
```

```
int main()
{
    string A, B;
    cin >> A >> B;
    cout << multi(A, B);
    return 0;
}
```

# 国王和麦粒



- 国王又来啦！这次要奖赏的是国际象棋的发明人。国际象棋盘有64个格子，第一个格子放一颗麦粒，第二个格子放两颗麦粒，第三个格子放四颗麦粒……依次类推
- 现在国王想知道他一共要准备多少颗麦粒才能放满国际象棋的棋盘

Sample input	Sample output
无输入	18446744073709551615

# 分析



- 最后一个格子的麦粒数为 $2^{63}$ ，这刚好超过long long的上限，那么求和只能使用高精度运算。
- 计算 $1+2+4+8+\cdots+2^{63}$
- 于是开心地写高精度加法了？

# 分析



- 打一个表， 不难发现：

i	$2^{i-1}$	$1+2+4+\dots+2^{i-1}$
1	1	1
2	2	3
3	4	7
4	8	15
5	16	31
6	32	63

- $1+2+4+\dots+2^{i-1}=2^i-1$

# 分析



- 于是这个问题转化为求 $2^{64}-1$
  - 所以相当于我们写一个高精度的pow(a, b)函数就行了
  - 又因为 $2^{62}$ 还在long long范围内
- 所以可以写高精度加法： $2^{62}+2^{62}+2^{62}+2^{62}-1$
- 或者写高精度乘法： $2^{62}*4-1$

三种不同的写法任君选择

# 参考代码

因为 $2^{62}$ 还在longlong范围内，也可以直接分解各位数字存入int数组



```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char A[100];
    int i, a[3000], c[3000];
    long long x = 1;
    for (int i = 1; i <= 62; i++) x *= 2;
    sprintf(A, "%lld", x);
    int Len = strlen(A);
    for (i = Len - 1; i >= 0; i--) a[Len-1-i] = A[i] - '0';
    for (i = 0; i < Len; i++) c[i] += a[i] * 4;
    for (i = 0; i < Len; i++)
        c[i+1] += c[i] / 10, c[i] = c[i] % 10;
    if (c[i]) i++;
    for (i = i - 1; i > 0; i--) cout << c[i];
    cout << c[0] - 1; // 个位数单独处理  $2^b-1$ 
    return 0;
}
```



# sprintf 函数

- sprintf 函数用于把数值转换为字符数组
- 其中占位符的用法和printf一致
- 其实可以把sprintf理解为printf, 只不过是输出到字符数组中

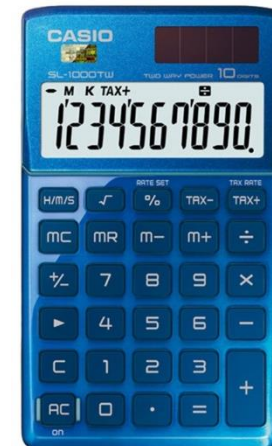
```
9223372036854775807
3.141593
-----
Process exited after 0.1192 seconds w
ith return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char s[100], s2[100];
    long long a = 9223372036854775807;
    float b = 3.1415926;
    sprintf(s, "%lld", a);
    sprintf(s2, "%f", b);
    cout << s << endl << s2;
    return 0;
}
```

# 高精度减法

- A-B problem

$$0 \leq A, B \leq 10^{10000}$$

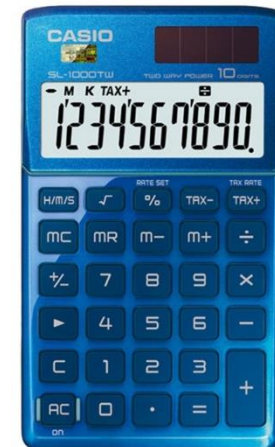


Sample input	Sample output
9223372034707292160 2019020190201902019	7204351844505390141

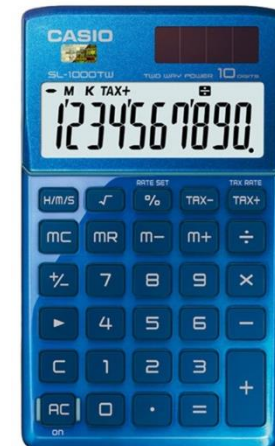
# 高精度减法

- 高精度减法和加法相类似，只是变进位为借位：  
如果不够减，就需要向高位借10，同时高位减1

```
if (a[j] < b[j])  
    a[j] += 10, a[j+1] --;
```



# 高精度减法

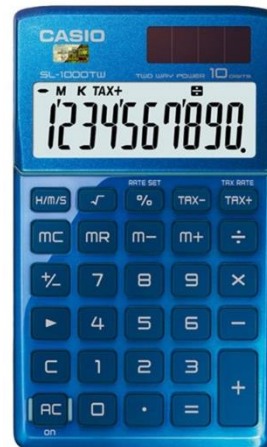


- 同时减法得到的差值有可能为负数，所以还需要特判但此时的两个运算数都是字符串形态，如何比较大小？
- 我们以前学习字符串的时候，知道字符串是可以直接比较大小的



```
if (A < B)
{
    C = A; A = B; B = C;
    cout << "-";    //先输出负号
}
```

# 字符串大小的比较



- 比如“123”和“124”这两个字符串， $123 < 124$ ，这与数值大小相符
- 但是“125”和“1221”这两个字符串， $125 > 1221$ ，这与数值大小不符

```
if (A.size() < B.size() || (A.size() == B.size() && (A < B)))  
{  
    C = A; A = B; B = C;  
    cout << "-";    //先输出负号  
}
```

# 参考代码

```
#include<bits/stdc++.h>
using namespace std;
string subtr(string A, string B)
{
    string ans;
    int i, j = 0, a[10010], b[10010], c[10010];
    int LA = A.size(), LB = B.size();
    if (LA < LB || ((LA == LB) && (A < B)))
    {
        ans = A; A = B; B = ans;
        ans = "-"; //处理差可能为负数
    }
    for (i = LA - 1; i >= 0; i --) a[LA-1-i] = A[i] - '0';
    for (i = LB - 1; i >= 0; i --) b[LB-1-i] = B[i] - '0';
    while (j < LA || j < LB)
    {
        if (a[j] < b[j]) {a[j] += 10, a[j+1] --;} //借位
        c[j] = a[j] - b[j];
        j ++;
    }
    while (!c[j] && j >= 1) j --; //清理高位零
    for (; j >= 0; j --) ans += c[j] + '0';
    return ans;
}
```

```
int main()
{
    string A, B;
    cin >> A >> B;
    cout << subtr(A, B);
    return 0;
}
```

# 结构体

- int 类型可以描述整型数，double类型可以描述浮点数，string可以描述字符串
- 但世间万事万物，有些数据单一类型无法描述，比如学生档案，就涉及姓名(string)、性别(char)、年龄(int)、学籍号(string)、考分(float)、是否获奖(bool)等

# 结构体

- 那我们希望有数据类型长这样：

姓名	学号
成绩	三好学生

- 然后数组长这样：

a[0]		a[1]		a[2]		a[3]		a[4]		a[5]	
姓名	学号	姓名	学号	姓名	学号	姓名	学号	姓名	学号	姓名	学号
成绩	三好学生	成绩	三好学生	成绩	三好学生	成绩	三好学生	成绩	三好学生	成绩	三好学生



# 结构体

- 我们知道除了系统中自带的函数以外，可以根据需要自定义函数
- 结构体实际上就是自定义复合数据类型

# 结构体的定义

- 定义结构体类型:
  1. 该数据类型名为student, 这个名称与int、float等名称同性质
  2. 每一个student类型包含四个域
- 继续定义元素类型为student的数组a:

```
struct student
{
    char name[20];
    char code[10];
    float score;
    bool excellent;
};
```

```
student a[1010];
```

# 结构体的定义

- 一般写成二合一：

```
struct student
{
    char name[20];
    char code[10];
    float score;
    bool excellent;
} a[1010];
```

# 结构体类型域的引用

- 在这个例子中，变量a、b都是student类型，要访问其score域，需要使用a.score

```
struct student
{
    char name[20];
    char code[10];
    float score;
    bool excellent;
} a, b;

b.score = a.score;
```

# 结构体的使用

```
#include<bits/stdc++.h>
using namespace std;
int n, cnt;
struct student
{
    char name[20];
    char code[10];
    float score;
    bool excellent;
} a[1010];
int main()
{
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        scanf("%s%s%f%d", a[i].name, a[i].code, &a[i].score, &a[i].excellent);
    for (int i = 0; i < n; i++)
        if (a[i].excellent)
        {
            printf("%s %s\n", a[i].name, a[i].code);
            cnt++;
        }
    printf("共有以上%d位同学入选", cnt);
    return 0;
}
```

```
5
张三 20190301 115 1
李四 20190302 97 0
王五 20190303 108 1
赵六 20190304 127 1
陈七 20190305 99 0
```

```
张三 20190301
王五 20190303
赵六 20190304
共有以上3位同学入选
-----
```

# 结构体的使用

- 从这个例子我们可以看出：  
结构体的最大意义在于把相关联的不同单一数据类型组合在一起

```
5
张三 20190301 115 1
李四 20190302 97 0
王五 20190303 108 1
赵六 20190304 127 1
陈七 20190305 99 0

张三 20190301
王五 20190303
赵六 20190304
共有以上3位同学入选
-----
```

# 谁拿了最多奖学金



- 某校的惯例是在每学期的期末考试之后发放奖学金。发放的奖学金共有五种，获取的条件各自不同：
- 院士奖学金，每人8000；五四奖学金，每人4000；成绩优秀奖，每人2000；西部奖学金，每人1000；班级贡献奖，每人850，详见luogu1051
- 只要符合条件就可以得奖。现在给出若干学生的相关数据，请计算哪些同学获得的奖金总数最高

Sample input	Sample output
4 YaoLin 87 82 Y N 0 ChenRuiyi 88 78 N Y 1 LiXin 92 88 N N 0 ZhangQin 83 87 Y N 1	ChenRuiyi 9000 28700

# 分析



- 把相关信息存入结构体数组后，在结构体数组中维护最值即可

```
struct node
{
    char name[21];    //姓名
    int pj;           //期末平均成绩
    int py;           //班级评议成绩
    char gb;          //学生干部
    char xb;          //西部省份
    int lw;           //论文
    int money = 0;    //该生所得奖金
} a[110];
```



# 参考代码

```
#include<bits/stdc++.h>
using namespace std;
int n, sum, maxm, pos;    //sum、maxm、pos 记录奖金总额、最高奖金额及序号
struct node
{
    char name[21];    //姓名
    int pj;    //期末平均成绩
    int py;    //班级评议成绩
    char gb;    //学生干部
    char xb;    //西部省份
    int lw;    //论文
    int money = 0;    //该生所得奖金
} a[110];
```

```
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%s%d%d%c%c%d", a[i].name, &a[i].pj, &a[i].py, &a[i].gb, &a[i].xb, &a[i].lw);
        if (a[i].pj > 80 && a[i].lw >= 1) a[i].money += 8000;    //院士奖学金
        if (a[i].pj > 85 && a[i].py > 80) a[i].money += 4000;    //五四奖学金
        if (a[i].pj > 90) a[i].money += 2000;    //成绩优秀奖
        if (a[i].pj > 85 && a[i].xb == 'Y') a[i].money += 1000;    //西部奖学金
        if (a[i].py > 80 && a[i].gb == 'Y') a[i].money += 850;    //班级贡献奖
        sum += a[i].money;
        if (a[i].money > maxm)
            pos = i, maxm = a[i].money;
    }
    printf("%s\n%d\n%d", a[pos].name, maxm, sum);
    return 0;
}
```

# Sort 函数

```
5
2 1 3 5 4
1 2 3 4 5
-----
Process exited after 4.179 seconds wi
th return value 0
请按任意键继续. . .
```

- Sort 函数用于给数组元素排序，默认从小到大

```
#include<bits/stdc++.h>
using namespace std;
int a[10010], n;
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    sort(a, a + n);
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    return 0;
}
```

# Sort 函数

- 如果要从大到小?

```
#include<bits/stdc++.h>
using namespace std;
int a[10010], n;
bool cmp(int a, int b)
{
    return a > b;
}
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    sort(a, a + n, cmp);
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    return 0;
}
```

```
5
2 1 3 5 4
5 4 3 2 1
-----
Process exited after 5.232 seconds wi
th return value 0
请按任意键继续. . .
```

# Sort 函数

- 这个写法:

```
return a > b;
```

- 等价于:

```
if (a > b) return true;  
else return false;
```

# Sort 函数

- 现实需求中常有更复杂的应用：比如足球比赛中，排名先看积分，积分相同时，则看净胜球/进球数/胜负关系等等；又比如成绩排名，以成绩为第一排序关键词，成绩相同的同学，学号小的靠前
- 我们现在以携带物品为例：优先携带价格高的，如果价格一样，则携带重量轻的

# Sort 函数

- 显然我们需要开结构体数据类型来表示一个物品

```
struct pack
{
    int value; // 价格
    int weight; // 重量
} a[10010];
```

# Sort 函数

```
#include<bits/stdc++.h>
using namespace std;
struct pack
{
    int value;
    int weight;
} a[10010];
bool cmp(pack x, pack y)
{
    if (x.value == y.value) return x.weight < y.weight; //等价值则重量轻在前
    return x.value > y.value;
}
int main()
{
    int n; scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d%d", &a[i].value, &a[i].weight);
    sort(a, a + n, cmp);
    for (int i = 0; i < n; i++)
        printf("%d %d\n", a[i].value, a[i].weight);
    return 0;
}
```

```
5
180 5 160 4 100 2 160 5 180 6

180 5
180 6
160 4
160 5
100 2
```

# Sort 函数

- 这个写法:

```
bool cmp(pack x, pack y)
{
    if (x.value == y.value) return x.weight < y.weight;
    return x.value > y.value;
}
```

- 等价于:

```
bool cmp(pack x, pack y)
{
    if (x.value > y.value) return true;
    else if (x.value == y.value)
    {
        if (x.weight < y.weight) return true;
        else return false;
    }
    return false;
}
```



# 分数线划定



- 为了选拔最合适的人才，A市对所有报名的选手进行了笔试，笔试分数达到面试分数线的选手方可进入面试。面试分数线根据计划录取人数的150%划定，即如果计划录取 $m$ 名志愿者，则面试分数线为排名第 $m \times 150\%$ （向下取整）名选手的分数，而最终进入面试的选手为笔试成绩不低于面试分数线的所有选手
- 现在就请你编写程序划定面试分数线，并输出所有进入面试的选手的报名号和笔试成绩（参加人数至多5000人）

Sample input	Sample output
6 3 //参加人数和录取人数	88 5 //分数线和面试人数
1000 90	1005 95
3239 88	2390 95
2390 95	1000 90
7231 84	1001 88
1005 95	3239 88
1001 88	

# 分析

- 开结构体数组存储每个考生的信息
- 注意处理好分数并列的情况就可以



# 参考代码

```
#include<bits/stdc++.h>
using namespace std;
struct node
{
    int number, score;
} a[5010];

int cmp (node x, node y)
{
    if (x.score == y.score) return x.number < y.number;
    return x.score > y.score;
}
```

```
int main()
{
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++)
        scanf("%d%d", &a[i].number, &a[i].score);
    m = floor(m * 1.5); //向下取整
    sort(a, a + n, cmp);
    int line = a[m-1].score; //分数线
    for (int i = m; i < n; i++)
        if (a[i].score == line) m++;
        else break; //处理分数并列的情况
    printf("%d %d\n", line, m); //分数线和人数
    for (int i = 0; i < m; i++)
        printf("%d %d\n", a[i].number, a[i].score);
    return 0;
}
```

# 期末成绩单



- 期末考试结束了，某门课程的任课老师要制作一份期末排名表。要求是这样的：成绩单以学号为序输出每个同学的成绩排名。但为了隐私，要求隐去其他信息（学生至多2000人，不考虑成绩并列）

Sample input	Sample output
5 //学生人数	4
98 //考试分数，下同	1
132	3
116	5
79	2
124	

index	score	rank
1	98	4
2	132	1
3	116	3
4	79	5
5	124	2

# 分析



- 如果用多个一维数组，排序时没法保持一一对应的关系
- 开结构体数组用于存储学生的信息

```
struct node
{
    int index, score, rank;
} a[2010];
```

index	score	rank

# 分析



- 读入时，不仅可以获得score，还要保存输入顺序index
- 注意为了和学号对应，我们从1开始

```
for (int i = 1; i <= n; i ++)  
{  
    scanf("%d", &a[i].score);  
    a[i].index = i;  
}
```

index	score	rank
1	98	
2	132	
3	116	
4	79	
5	124	

# 分析



- 然后按score排序一次，获得rank
- 再按index排序一次，得到（最终的成绩单）输出顺序

index	score	rank
1	98	4
2	132	1
3	116	3
4	79	5
5	124	2

# 参考代码

这种操作也可以  
称为“原地排序”  
怎么实现？



```
#include<bits/stdc++.h>
using namespace std;
struct node
{
    int index, score, rank;
} a[2010];
bool cmpscore(node x, node y) //按成绩排序
{
    return x.score > y.score;
}
bool cmpindex(node x, node y) //按学号排序
{
    return x.index < y.index;
}
```

```
int main()
{
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &a[i].score);
        a[i].index = i;
    }
    sort(a + 1, a + n + 1, cmpscore);
    for (int i = 1; i <= n; i++)
        a[i].rank = i; //此时的a[]已按score排好序
    sort(a + 1, a + n + 1, cmpindex);
    for (int i = 1; i <= n; i++)
        printf("%d\n", a[i].rank);
    return 0;
}
```



# 文件操作

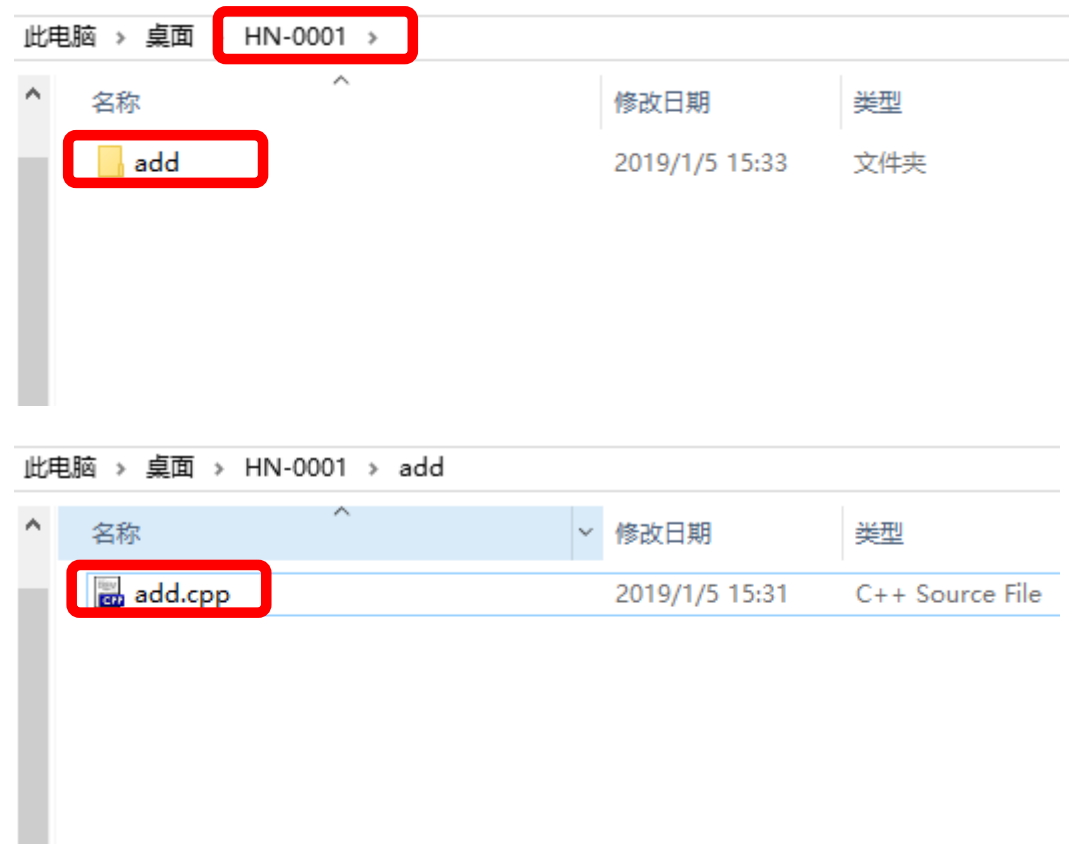
- 我们通过重定向操作实现文件读写

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    freopen("add.in", "r", stdin); //标准写入
    freopen("add.out", "w", stdout); //标准输出
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d", a + b);
    fclose(stdin); //关闭stdin
    fclose(stdout); //关闭stdout
    return 0;
}
```

# 文件操作

1. 考生个人目录，名字为考生考号
2. 为每道题建立对应的英文子目录
3. 把对应英文名的源程序cpp放入其中
4. 写好对应文件重定向语句

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    freopen("add.in", "r", stdin);
    freopen("add.out", "w", stdout);
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d", a + b);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```



# 真实案例

## 湖南广益实验中学信息学奥赛培训班期末测试

(七年级上学期)

题目名称	奇数偶数	珠心算	国王和骑士	游戏
题目类型	传统型	传统型	传统型	传统型
目录	odd	count	knight	game
可执行文件名	odd	count	knight	game
输入文件名	odd.in	count.in	knight.in	game.in
输出文件名	odd.out	count.out	knight.out	game.out
每个测试点时限	1.0 秒	1.0 秒	1.0 秒	1.0 秒
内存限制	128MB	128MB	128MB	128MB
测试点数目	10	10	10	10
每个测试点分值	10	10	10	10

名称	修改日期	类型
count	2019/1/5 15:44	文件夹
game	2019/1/5 15:44	文件夹
knight	2019/1/5 15:44	文件夹
odd	2019/1/5 15:44	文件夹

# getchar函数/putchar函数

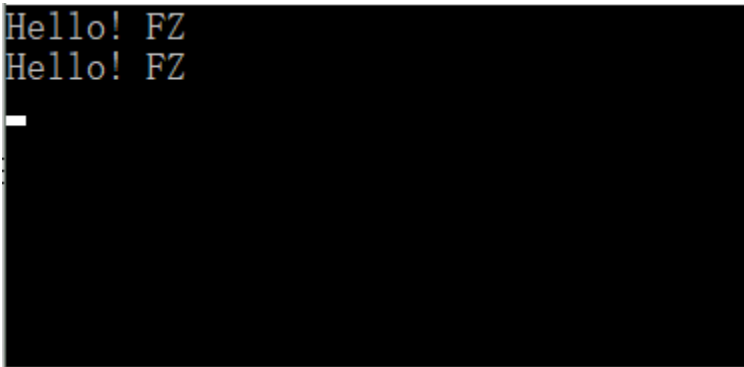
- 这是一对函数：**getchar()** / **putchar()**
- 用于读入/输出单个字符

```
Hello
H
-----
Process exited after 10.46 seconds with return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char c;
    c = getchar();
    putchar(c);
    return 0;
}
```

# getchar函数/putchar函数

- 这是一对函数: **getchar()** / **putchar()**
- 或者这样用:



```
Hello! FZ
Hello! FZ
_
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char c;
    while (c = getchar())
        putchar(c);
    return 0;
}
```

# 快速读入

- 本来到此为止了： `getchar/putchar`这都是很正经的函数
- 但是： `getchar()`函数的读入速度， 比`scanf`还要快！
- 于是常常被TLE折磨的Oier， 就想到要利用`getchar`的快速

# 快速读入

本页PPT为选学，  
不要求掌握



- 于是有了这段快速读入代码

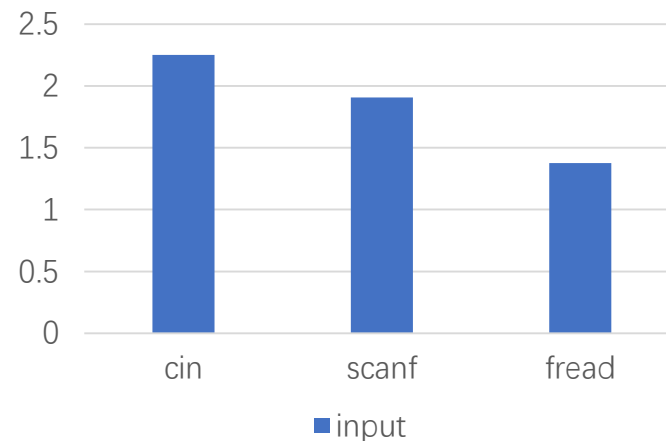
```
int read() //也可以根据题意定义 Long Long 型
{
    char ch = getchar(); int num = 0, f = 1;
    while (ch < '0' || ch > '9') {if (ch == '-') f = -1; ch = getchar();} //处理负数
    while (ch >= '0' && ch <= '9') {num = num * 10 + (ch - '0'); ch = getchar();}
    return num * f;
}

n = read(); // main函数中调用 read()
```

# 快速读入

- 我们可以看到： $10^6$ 规模的数据输入，快速读入比scanf要快约40%，比cin要快约65%

试题		选手				
名称	排名	input	总分	总用时(s)	测试时间	
fread	1	100	100	1.375	2019/1/8 17:33:47	
scanf	1	100	100	1.906	2019/1/8 17:23:56	
cin	1	100	100	2.25	2019/1/8 17:23:47	





# 快速读入

- 但这并不是终极的fread版本，因为大多数情况下，它比scanf快得有限
- 终极fread适合卡常到极致的选手，有需要者可自行百度
- 搭配快速输出fwrite食用更美妙

# 二进制

- 我们都知道，计算机采用二进制存储数据
- 进一步的，二进制数在计算机中是以补码形式存储的

# 补码

- 计算机中的码制有原码、反码、补码三种
- 比如10，二进制表示为：00001010（一个字节8位）
- 注意前面那个最高位的0，是符号位，0表示正数
- 那么-10的二进制表示呢？不是写成-00001010
- 而是写成：10001010，也就是最高位的符号位1表示负数

# 补码

- 所以，一个字节能表示多大的数？  
 $-127 \sim 127$  ( $-2^7 \sim 2^7$ )
- 而int是四个字节存储，所以能表示  
 $-2^{31}-1 \sim 2^{31}-1$

# 补码

- 计算机中的码制有原码、反码、补码三种
- 这里的00001010 (10) 以及10001010 (-10) 都是原码
- 反码即是如果符号位为1, 则除最高位符号位以外, 按位取反
- 10的反码表示是00001010
- -10的反码表示是11110101

# 补码

- 计算机中的码制有原码、反码、补码三种
- 这里的00001010 (10) 以及10001010 (-10) 都是原码
- 这里的00001010 (10) 以及11110101 (-10) 都是反码
- 补码即是如果符号位为1, 则除最高位符号位以外, 反码+1
- 10的补码表示是00001010
- -10的补码表示是11110110

# 补码

- 计算机中的码制有原码、反码、补码三种
- 这里的00001010 (10) 以及10001010 (-10) 都是原码
- 这里的00001010 (10) 以及11110101 (-10) 都是反码
- 这里的00001010 (10) 以及11110110 (-10) 都是补码
- 计算机中的二进制存储都是采用的补码

# 补码

- 为什么都要采用补码呢？

- 我们试试看 $10-10$ ： $10+(-10)$

$00001010 + 11110110 = 00000000$  （注意符号位也是参与运算的！）

- 所以现在可以回答为什么计算机中的二进制存储要用补码：
- 为了把所有的加减法统一成加法



# 无符号数

- 这里就顺便介绍一个新概念：无符号数：**unsigned**
- 现在我们就可以理解了：无符号数就是字节的最高位也表示具体的数值而不再表示数的正负
- 它的好处是可以容纳的范围比有符号数大一倍，坏处是不能再表示负数了

# 无符号数

- 2147483648原本定义为int是要爆的
- 如果某些题明确是非负的，就可以定义为无符号数

2147483648

-----  
Process exited after 0.1927 seconds with return value 0  
请按任意键继续. . .

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    unsigned int n = 2147483648;
    printf("%u", n);
    return 0;
}
```

# 位运算： 移位

- 这两个程序得到的结果都是10， 但是位运算更快
- 原因是位运算是直接对二进制操作， 省却了中间转换十进制的过程
- 二进制移位运算： <<、>>

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 20;
    printf("%d", n / 2);
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 20;
    printf("%d", n >> 1);
    return 0;
}
```

# 高低位交换



- 给一个正整数 $n$ 。这个数可以用一个32位的二进制数表示（不足32位补0）。我们称这个二进制数的前16位为“高位”，后16位为“低位”。将它的高低位交换，我们可以得到一个新的数。问这个新的数是多少（用十进制表示）

$n \in \text{int}$

Sample input	Sample output
1314520	249036820

# 分析



- 这个题暴力也能做，因为 $2^{16}=65536$ ，所以可以通过对65536取整、取模分离出前后16位
- 不过用位运算更快捷

# 参考代码

注意这里如果不用无符号数，可能会爆的，为什么？



```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    unsigned int n;
    scanf("%u", &n);
    printf("%u", (n >> 16) + (n << 16));
    return 0;
}
```

# 位运算： 按位与/按位或

- 这两个程序得到的结果都是“odd”， 同样位运算更快
- 二进制按位与运算： **&**， 按位或运算： **|**

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 101;
    if (n % 2 == 1) printf("odd");
        else printf("even");
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 101;
    if (n & 1) printf("odd");
        else printf("even");
    return 0;
}
```

# 有多少个1

- 给一个正整数 $n$ ，问它的二进制表示中有多少个1

Sample input	Sample output
15	4



# 位运算

- $x \& x - 1$ 可以快速去掉 $x$ 的二进制位中最后一个1
- 顺便提一下：位运算的优先级别相当低：比加减还低

```
4
-----
Process exited after 0.1192 seconds w
ith return value 0
请按任意键继续. . .
```



```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 15, cnt = 0;
    while (n > 0)
        cnt ++, n &= n - 1;
    printf("%d\n", cnt);
    return 0;
}
```

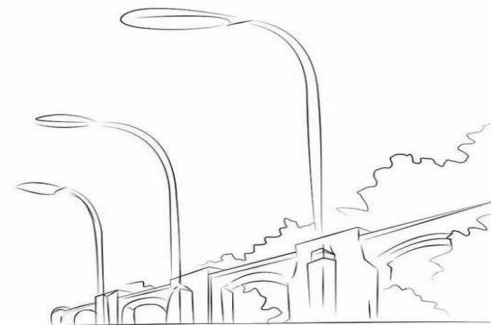
# 位运算： 按位异或

- 二进制按位异或运算： ^， 一个数与0异或得它自己， 与它自己异或得0

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 101;
    printf("%d\n", n ^ 0);
    printf("%d\n", n ^ n);
    return 0;
}
```

```
101
0
-----
Process exited after 0.1179 seconds with return value 0
请按任意键继续. . .
```

# 路灯



- 道路上有很多盏路灯，编号1、2、3、4……，每盏灯只有打开和关闭两种操作，原本所有的灯都是关闭的。现在有个人走过，每次做这样的操作：将编号为 $a$ 、 $2a$ 、 $3a$ 、…… $ka$ 的灯各按一次开关。
- 在进行了 $n$ 次操作后，发现只有一盏灯是亮的，问那盏灯的编号是多少

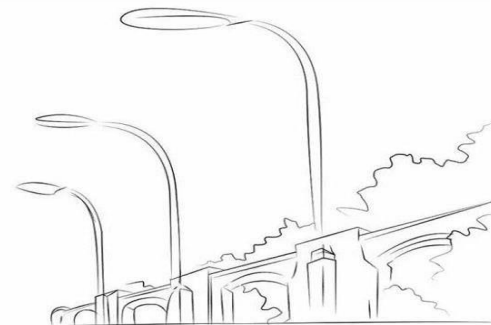
Sample input	Sample output
3 //n 1 4 //a, k 2 5 3 2	3

# 路灯



- 按一次开关的意思是：原本是关的会打开，原本是开的会关闭
- 一个灯原本是关闭的，如果被操作偶数次，必然还是关闭
- 也就是说，最后亮的那一盏，必然是被操作奇数次的

# 路灯



- 得到这个结论，其实我们模拟也可以做了
- 异或运算有一个特性：一个数异或同一个数两次，答案保持不变
- 为什么？
- 用异或运算比模拟一遍更快捷：
- 我们直接把所有符合条件的编号异或起来
- 那些出现偶数次的编号必然被抵消掉了
- 异或运算到最后，得到的自然就是那个出现奇数次的

# 参考代码

```
#include<bits/stdc++.h>
using namespace std;
int n, a, k, ans;
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i ++)
    {
        scanf("%d%d", &a, &k);
        for (int j = 1; j <= k; j ++)
            ans ^= a * j; // 利用异或的特性
    }
    printf("%d", ans);
    return 0;
}
```

# 课外加练

- luogu 2142      高精度减法
- luogu 1045      麦森数
- luogu 1051      谁拿了最多奖学金
- luogu 1068      分数线划定
- luogu 1100      高低位交换
- luogu 1161      开灯
- luogu 1461      海明码