带余除法

出这道题的目的只是让你们学会题面里描述的 C/C++ 除法和取模对负数的情况。

请各位选手牢牢记住这两条性质：**负数整除向零靠近，取模符号与被除数符号相同。**以后会用上 ~ 。°·(ノ/>/︿/</ノ)·°。

```
#include <bits/stdc++.h>
using namespace std;
int a, b;
signed main()
{
    scanf("%d%d", &a, &b);
    printf("%d * %d + %d = %d", a / b, b, a % b, a);
    return 0;
}
```

区间

这题其实就是除法运算的应用题。

根据上一题，我们知道，整除 `y/x` 满足 `(y / x) * x + y % x = x`，若 x, y 都是非负整数且 $x \neq 0$，则 `y/x` $= \lfloor \frac{y}{x} \rfloor$ 即除法下取整，且 $y \bmod x \geq 0$。放缩得：$\lfloor \frac{y}{x} \rfloor x \leq \lfloor \frac{y}{x} \rfloor x + (y \bmod x) = x$，也就是说 $\lfloor \frac{y}{x} \rfloor x$ 一定是左端点 u，则 $\lfloor \frac{y}{x} \rfloor x + x$ 是右端点 v。

若 $x < 0$，不妨直接取 $x = |x|$，不会影响答案。因为一个正数的所有倍数同时也是它的相反数的倍数。即如 3 的倍数可以有 $\cdots, -6, -3, 0, 3, 6, \cdots$，那么 -3 的倍数也可以有 $\cdots, -6, -3, 0, 3, 6, \cdots$，故 $\pm x$ 是等效的。

根据上一题，若 $y < 0$，则我们发现有 `y/x` $= \lceil \frac{y}{x} \rceil$ 即除法上取整，那么若 $y \bmod x \neq 0$，必然满足 $\lfloor \frac{y}{x} \rfloor = \lceil \frac{y}{x} \rceil + 1$。否则，若 $y \bmod x = 0$，则$y = \lceil \frac{y}{x} \rceil = \lfloor \frac{y}{x} \rfloor$ 同时在两个区间 $y \in [\lceil \frac{y}{x} \rceil x, \lceil \frac{y}{x} \rceil x + x], y \in [\lceil \frac{y}{x} \rceil x - x, \lceil \frac{y}{x} \rceil x]$，显然后者更小。因此，无论如何，都可以当 $y < 0$ 或 $y \bmod x = 0$ 时直接取 $u = \lceil \frac{y}{x} \rceil - 1$。

因此答案为：$u = \lfloor \frac{y}{|x|} \rfloor - [y < 0 \; or \; (y \bmod x = 0)]$。其中 $[st]$ 是逻辑表达式，若 st 为真取值为 1，否则取值为 0。且 $v = u + |x|$。

显然，$|x| > 0$，即 C99/C++11 的 `y/abs(x)` 总是等于 $\lfloor \frac{y}{|x|} \rfloor$。

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
ll x, y, k;
signed main()
{
    scanf("%lld%lld", &x, &y), x = abs(x);
    k = y / x - (y <= 0 || y % x == 0);
    printf("%lld %lld", k * x, k * x + x);
    return 0;
}
```

pow

多数编程语言库函数自带的 pow 函数是浮点数计算。众所周知，浮点数会产生误差。所以，请手写实现整数的 pow。

如果您非要用 pow，请使用 long double。不管是 float 还是 double 精度都不足。

数据范围参见下表：

- `int` 通常 $[-2^{31}, 2^{31}-1]$，4 字节，其中 $2^{31} \approx 2.1 \times 10^9$
- `long long` $[-2^{63}, 2^{63}-1]$，8 字节，其中 $2^{63} \approx 9.2 \times 10^{18}$
- `unsigned long long`，$[0, 2^{64}-1]$，8 字节
- `__int128_t` $[-2^{127}, 2^{127}-1]$，16 字节，其中 $2^{127} \approx 8.5 \times 10^{37}$
- `double` 有效位数约 15 位，8 字节，范围约为 $[-1.79 \times 10^{308}, 1.79 \times 10^{308}]$
- `long double` 通常有效位数约 20 位，16 字节，范围约为 $[-1.2 \times 10^{4932}, 1.2 \times 10^{4932}]$

参考代码：(**强烈建议使用第一种**，而不是第二种即 `pow` 函数)

> 虽然在本题使用浮点数误差并不大，但是如果在以后的做题里，需要连续进行大量的浮点数运算，那么这些误差累计起来，误差范围会指数级增长，因此，慎用浮点数，需要牢记浮点数误差。

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long; //即 #define ll long long
ll x, y, r = 1;
signed main()
{
    scanf("%lld%lld", &x, &y);
    for (ll i = 0; i < y; ++i)
    {
        r *= x;
    }
    printf("%lld", r);
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
ll x, y;
signed main()
{
    scanf("%lld%lld", &x, &y);
    printf("%.0Lf", pow(1.0L * x, y));
    return 0;
}
```

数据范围

参考上一题的解析，可知 unsigned long long 和 long double 都可以过(当然整数运算我们**强烈建议使用 ull**)

```cpp
#include <bits/stdc++.h>
using namespace std;
using ull = unsigned long long;
ull x, y;
signed main()
{
    scanf("%llu%llu", &x, &y);
    printf("%llu", x * y);
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
using db  = long double;
db x, y;
signed main()
{
    scanf("%Lf%Lf", &x, &y);
    printf("%.0Lf", x * y);
    return 0;
}
```

思考题：可以使用 `__int128_t`。注意到 i128 的输入和输出是未定义的，需要自行手写输入输出，如：

```cpp
#include <bits/stdc++.h>
using namespace std;
using ull = __int128_t;
ull x, y, r;
//注意这里的 input, print 只能处理非负数，负数请自行修改
void input(ull &v) //传引用语法,能起到指针的效果
{
    char s[30] = {};
    scanf("%s", s);
    v = 0;
```

```
    for (int i = 0; s[i]; ++i)
    {
        v = v * 10 + (s[i] - '0');
    }
}
void print(ull v)
{
    if (v > 10)
    {
        print(v / 10);
    }
    putchar('0' + (v % 10));
}
signed main()
{
    input(x);
    input(y);
    r = x * y;
    print(r);
    return 0;
}
```

当然您也可以用高精度来做本题，但是没必要。感兴趣自行到 OJ 题库搜索高精度，能找到相关模板题。

星月选课

题解

星月学语

题解

星月观星

题解

星月寻人

题解