高精度运算

突破整数位数上限：倒序存位与竖式四则运算

HGNU-ACM 实验室算法组

配套：倒序存储原理 + 高精加减乘除 + 5 道一本通真题精讲





本章目录（前半）

为什么 long long 装不下与倒序存储

高精度加法与进位传播

高精度减法、大小比较与借位

高精度乘单精度与高精度乘法

承接第 2 章《排序与逆序对》：当数字大到连 `long long` 的 19 位也装不下时，先学倒序存位对齐，再逐个攻克高精加、减、乘。

本章目录（后半）

高精度除以单精度与余数

五道高精度一本通真题精讲

常见错误、检查清单与练习安排

后半章掌握正序高精除以单精度、5 道一本通经典真题（含 NOIP 回文数）实战，以及检查清单。

痛点引入：long long 的天花板在哪里？

第 2 章算 $n = 10^5$ 的逆序对时，结果 5×10^9 撑爆了 int，我们换用 long long 解决了问题。可如果题目要算 2^{100} 或两个 200 位数相加呢？

整数类型	最大值上限	最多几位数	超出后会发生什么？
int	约 2.14×10^9	10 位	溢出成负数或错误数值（**WA**）
long long	约 9.22×10^{18}	19 位	算 2^{100}（31 位）直接溢出 **WA**
double	约 10^{308}	只保 15 位有效数字	低位数字全部被四舍五入丢掉（**WA**）

- C++ 里没有任何内置基本类型能精确保存 200 位整数！

本页要点

当整数位数超过 18 位且要求每一位都精确无误时，必须动用“高精度运算”。

大白话定义：什么是高精度？怎么存进“小盒子”？

生活类比：一节车厢只坐一个数字

既然一个整型变量盒子装不下 200 位数，我们就准备一列长 200 的整数数组 `vector<int>`，每个格子只存 $0 \sim 9$ 一位数字，然后用程序模拟小学列竖式！

- 读入第一步：绝对不能写 `long long x; cin >> x;`，必须先用字符串 `string s;` 把这串数字原封不动当文字读进来。
- 字符转数字：字符 `'5'` 在电脑里存的是 ASCII 编码 53，减去字符 `'0'`（编码 48）才真正变成整数 5，即 `s[i] - '0'`。
- 新术语——高精度运算：用数组存放超大整数的每一位，再通过循环模拟小学加减乘除竖式来保证精确度。

本页要点

读大数永远用 `string`，拆位放进 `vector<int>` 时别忘了减去 `'0'`！

TikZ 图解：为什么要“倒序存储”来对齐竖式？

把字符串 "985" 存入数组 A，哪一端放在下标 0 决定了竖式能不能对齐：



- 小学竖式要求右对齐（个位对个位、十位对十位）。倒序存入数组后，不论两数有多长，A[0] 和 B[0] 永远都是个位！
- 打印结果时再从 C.size() - 1 倒着输出回 0 即可。

本页要点

核心约定：A[0] 存个位（最低位），A.back() 存最高位；输出时倒着打印回来！

手算跟踪表：字符串 "0035" 倒序读入与去前导零

有时输入或减法结果的高位会出现多余的 0。看 "0035" 倒序放入 C 后怎么用 pop_back 清理：

步骤	数组 C 内容	条件 C.size()>1 && C.back()==0	说明
读入	[5, 3, 0, 0]	进入循环前	原串最高位 0 跑到了末尾
第 1 轮	[5, 3, 0]	真（长度 $4>1$，末位 0）	弹出末尾第 1 个多余的 0
第 2 轮	[5, 3]	真（长度 $3>1$，末位 0）	弹出末尾第 2 个多余的 0
第 3 轮	[5, 3]	假（末位 $3\neq 0$）	退出循环，倒序输出得 35

本页要点

因为倒序把最高位放到了数组末尾，所以删前导零变成了极高效的 C.pop_back()！

正确性不变量：为什么必须写 C.size() > 1？

不变量阶段	要验证的内容
初始化	倒序读入后，字符串右起第 i 个字符落在 A[i]，代表十进制位权 10^i
保持	每次 pop_back() 只删除末尾（最高位）的 0，不改变数本身的值，且保证数组非空
终止	当最高位非 0 或长度只剩 1 时停下，若数值真为 0，最后恰好留住唯一一格 [0]

为什么不能写成 !C.empty()？

如果结果刚好等于 0（例如 000 或 123 - 123），写成 !C.empty() 会把最后一格 0 也一起删光，屏幕什么都输出不出来！

本页要点

写去前导零只有一句标准咒语：

```
while (C.size() > 1 && C.back() == 0)
C.pop_back();
```

大整数倒序读入、去零与输出模板

```
string s;
cin >> s;
vector<int> A;
for (int i = (int)s.size() - 1; i >= 0; --i) {
    A.push_back(s[i] - '0');                          // 从末尾到开头倒序存入
}
while (A.size() > 1 && A.back() == 0) {
    A.pop_back();                                      // 清除末尾多余前导 0
}
for (int i = (int)A.size() - 1; i >= 0; --i) {
    cout << A[i];                                      // 从最高位倒着打印回屏幕
}
```

本页要点

循环下标 `i` 减到 0 时务必使用有符号 `int`（不要写 `size_t`），否则 `0 - 1` 会下溢变超大整数死循环！

存储与去零上机验证（3 组数据 + 反事实追问）

把上面读写去零代码放进完整程序运行，用三组数据验证结果：

- 普通带前导零：输入 `0035` $\rightarrow$ 预期输出 `35`。
- 全零特殊边界：输入 `000` $\rightarrow$ 预期输出 `0`（留下最后一格 0）。
- 超长整数：输入 `987654321098765432101234 5` $\rightarrow$ 原样输出 25 位数不失真。

反事实追问

如果把倒序打印循环里的 `int i = (int)A.size() - 1` 错写成了 `size_t i = A.size() - 1; i >= 0; --i` 会发生什么？
答：无符号整数 `size_t` 永远 ≥ 0，`i = 0` 再 `--i` 会变成超大整数继续访问越界内存，触发死循环或段错误 **RE**！

存储与去零易错对照

错误类型	常见错误写法	正确做法与原因
WA（输出乱码）	`A.push_back(s[i])` 忘减 `'0'`	存成了 ASCII 码 $48 \sim 57$，必写 `s[i] - '0'`
WA（输出空行）	`while (!C.empty() && C.back()==0)`	算成 0 会删光全部元素，必写 `C.size() > 1`
RE（无限循环）	倒序循环写 `for (size_t i=...; i>=0; --i)`	倒着减到 0 必须把下标写成有符号 `int`
WA（先读溢出了）	先 `cin >> long long` 再转字符串	读入那一瞬间已经溢出，一开始就用 `string`

本页要点

记住“`string` 读、`int i` 倒序、减 `'0'` 存、`size()>1` 去零”四件套！

生活引入：小学竖式怎样算 999 + 1？

场景：把两个超长数字像小学算术一样对齐相加

不管是算 $999+1$，还是算两个 200 位数相加，规则其实只有一句：从个位（下标 0）开始对齐，每位加上上一位的进位，满 10 就留余数、向下一位进 1。

- 两数相加时，如果两个数位数不一样长（例如 999 有 3 位，1 只有 1 位），怎么办？
- 很简单：在短的那个数前面（也就是倒序数组的末尾更高位）用 0 补齐！代码里只要判断 `if (i < B.size()) s += B[i];` 就自动补好 0 了。

本页要点

倒序存储让下标 i 天然对齐：只需注意两个细节——“短数高位补 0”和“算完别丢最后进位”。

痛点与术语：如果高位下标越界或漏了进位怎么办？

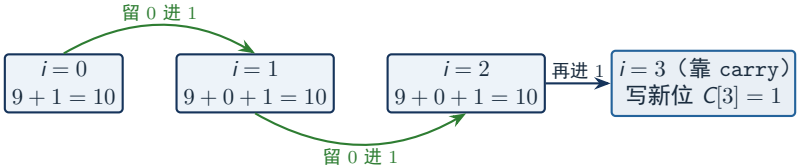
- 新术语——进位变量 `carry`：记录第 $i-1$ 位满十后带给第 i 位的进位（两个 $0 \sim 9$ 的数字再加进位，和最大为 $9+9+1=19$，所以 `carry` 永远只可能是 0 或 1）。
- 痛点一（越界 **RE**）：当 A 长 200 位、B 长 5 位时，循环跑到 $i=10$ 如果直接写 `A[i] + B[i]`，访问 `B[10]` 必定数组越界 **RE**！
- 痛点二（漏最高位 **WA**）：算 $999+1$ 时，A, B 都只有 3 位，如果循环只写 `i < max(A.size(), B.size())`，算完 $i=2$ 就停了，最高位多出来的 1 被丢掉，答案变成错误的 `000`！

本页要点

循环继续的条件必须写成 `i < A.size() || i < B.size() || carry`，三位一体缺一不可！

TikZ 图解：999 + 1 倒序数组逐位进位链路

以 $A=[9,9,9]$（代表 999）加上 $B=[1]$（代表 1）为例，看进位如何一路冲到新格子 $C[3]$：



- 当 $i=3$ 时，虽然 $i<|A|$ 和 $i<|B|$ 都已经为假，但因为 `carry == 1`，循环再多跑一轮，把这最后的 1 塞进 `C[3]`！

本页要点

靠 `||` `carry` 多跑的最后一轮，专门负责接住最高位产生的“新进位 1”。

手算跟踪表：999 + 1 的全部循环轮次

初始 $A=[9,9,9]$，$B=[1]$，$carry=0$，$C=[]$，跟踪每轮条件 `i<3 || i<1 || carry`：

i	循环条件成立？	$s=carry+A[i]+B[i]$	写 `C[i]`	新 `carry`	当前数组 C
0	真（$0<3$）	$0+9+1=10$	0	1	`[0]`
1	真（$1<3$）	$1+9+0=10$	0	1	`[0, 0]`
2	真（$2<3$）	$1+9+0=10$	0	1	`[0, 0, 0]`
3	真（`carry=1`）	$1+0+0=1$	1	0	`[0, 0, 0, 1]`
4	假（全假退出）	不执行	-	0	倒打得 **1000**

本页要点

第 3 轮把 `carry` 消耗归零，第 4 轮三个条件同时为假，循环干净退出！

正确性证明：高精度加法的循环不变量

不变量阶段	要验证的内容
初始化	$i = 0$, $carry = 0$，已处理 0 位，低 0 位数值和严格为 0
保持	第 i 轮加上 $A[i], B[i]$（越界按 0）和旧 carry 得 s，$s \bmod 10$ 写进第 10^i 位，$\lfloor s/10 \rfloor$ 变成新 carry 传给 10^{i+1} 位，总值严格守恒
终止	当 $i \geq \max(\lvert A \rvert, \lvert B \rvert)$ 且 $carry = 0$ 时退出，此时 C 的十进制数值精确等于 $A + B$

如果输入带了前导零呢？

比如输入 "005" 和 "007"：高位多加几轮 $0 + 0 = 0$ 写入 C 末尾，最后统一用 `while (C.size() > 1 && C.back() == 0) C.pop_back();` 就能一次清干净！

高精度加法核心模板代码

```cpp
vector<int> add_big(const vector<int> &A, const vector<int> &B) {
    vector<int> C;
    int carry = 0;
    for (size_t i = 0; i < A.size() || i < B.size() || carry; ++i) {
        int s = carry;                          // 继承上一位进位
        if (i < A.size()) s += A[i];            // 加 A 的第 i 位
        if (i < B.size()) s += B[i];            // 加 B 的第 i 位
        C.push_back(s % 10);                    // 留余数，满十进 1
        carry = s / 10;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

本页要点

加完顺手去前导零，就能一并兼容“输入自带前导零”和“0+0”！

高精度加法上机验证（3 组数据 + 反事实追问）

使用完整程序 `code/big_add.cpp` 在本地验证三组必测样例：

- 最小全零：输入 0 和 0 → 预期输出 0。
- 连续进位增位：输入 999 和 1 → 预期输出 1000。
- 输入带前导零：输入 005 和 007 → 预期输出 12。

反事实追问

如果把循环条件 `i < A.size() || i < B.size() || carry` 删掉最后的 `|| carry`，输入 999 和 1 会输出什么？

答：第 3 轮不会执行，最后产生的最高位进位 1 被永久遗忘，程序将输出错误答案 000（或去零后输出 0，严重 **WA**）！

高精度加法易错对照

错误类型	常见错误写法	正确做法与原因
WA（丢最高位）	循环条件忘写 `|| carry`	$999+1$ 等增位进位会漏掉最高位 1
RE（数组越界）	直接写 `s = carry + A[i] + B[i]`	两数位数不同时短数组会越界，必加 `if (i < size())`
WA（留前导零）	加完没有 `pop_back` 去前导零	输入含 005 时会错输出 012
WA（商余写反）	写 `C.push_back(s / 10); carry = s % 10;`	记住“余数留下，商进给高位”：`% 10` 留下，`/ 10` 进位

本页要点

加法写完必测三组：0+0、999+1、005+007！

生活引入：计算 1000 - 1 与 12 - 100 的痛点

减法的两个新麻烦

加法随便怎么加都不会出现负数位；但做减法 $A - B$ 时会遇到两个新难题：

1. 连续借位：算 $1000 - 1$，个位 0 不够减 1 要借位，十位也是 0、百位还是 0，一直借到千位！
2. 小减大出负号：如果输入是 $12 - 100$，直接按位往下减，到最高位借不到数，数组里就会出现负数，竖式彻底崩掉！

- 怎么化解小减大？遇到 $12 - 100$ 时，你在草稿纸上也是先在答案前面写一个负号 '-'，然后照样算 $100 - 12 = 88$，合起来就是 -88！

本页要点

高精度减法核心口诀：先比大小，保证永远只算“大减小”；遇到 $A < B$ 就先打印负号再交换！

新术语与通解策略：怎么比较两个倒序大整数谁大？

先清理输入可能带有的前导零后，怎么写比较函数 `less_vec(A, B)` 判断 $A < B$？

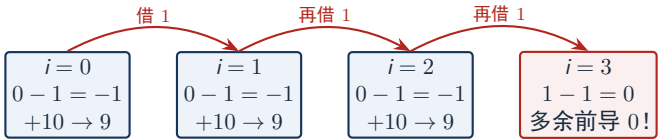
- 第 **1** 关（比位数长度）：正整数（无前导零）只要位数不一样，位数少的一定更小（比如 $99 < 100$）！直接返回 `A.size() < B.size()`。
- 第 **2** 关（位数相同，从最高位往下比）：记住我们的数组是倒序存的，所以最高位在 `A.size() - 1`！要从高位往低位找到第一个不相等的数字，谁小谁就小。
- 全相等时：循环走完都没分出胜负，说明 $A == B$，$A < B$ 不成立，返回 `false`！

本页要点

倒序数组比大小必须从下标 `size() - 1` 倒着往 `0` 比，千万不要从个位 `0` 开始比！

TikZ 图解：1000 − 1 连续借位与高位清零

已知 $A=[0,0,0,1]$（代表 1000），$B=[1]$（代表 1），观察借位信号 borrow 的传递：



- 算完后得到数组 $C=[9,9,9,0]$，最高位消成了 0，所以做完减法必须去前导零，弹出末尾的 0 得到 999！

本页要点

算减法不需要真的去改原数组 A，只要带着一个 borrow（0 或 1）往下传就行！

手算跟踪表：1000 − 1 逐位借位跟踪

设 $A=[0,0,0,1]$，$B=[1]$，$borrow=0$，循环跑 $i=0\sim3$（因为 $A\geq B$，位数绝不会超过 $|A|$）：

i	$A[i]$	$d=A[i]-borrow-B[i]$	够减吗？	写 C[i]	新 borrow 与说明
0	0	$0-0-1=-1$	不够（+10）	9	1（向第 1 位借 1）
1	0	$0-1-0=-1$	不够（+10）	9	1（连续借位）
2	0	$0-1-0=-1$	不够（+10）	9	1（继续借）
3	1	$1-1-0=0$	够减（≥0）	0	0（清零，退位完成）
4	结束	假（$i<4$ 为假）	退出循环	去零得 $[9,9,9]$	倒打结果 **999**

本页要点

第 3 步够减时 borrow 必须立刻复位成 0；算完后必须 pop_back 清理消出来的 0！

正确性证明：为什么“大减小 + 复位 borrow”一定对？

不变量阶段	要验证的内容
初始化	经 `less_vec(A, B)` 交换后确保 $A \geq B$，此时 $\lvert A\rvert \geq \lvert B\rvert$ 且初始 `borrow = 0`
保持	第 i 轮若 $d < 0$，则加 10 使 $C[i] \in [0, 9]$ 并向 10^{i+1} 位借 1（`borrow=1`）；若 $d \geq 0$，则直接写入 $C[i]$ 且复位 `borrow=0`
终止	因为 $A \geq B$，运行到最高位 $\lvert A\rvert - 1$ 后绝不会仍有 `borrow == 1`，去前导零后 C 精确等于 $A - B$

特别提醒

如果 A 和 B 的输入可能带前导零（如 005 和 9），必须先清除 A, B 的前导零再调用 `less_vec`，否则长度判断会误判！

模板代码 (1/2)：大整数比较函数 less_vec

```cpp
bool less_vec(const vector<int> &A, const vector<int> &B) {
    if (A.size() != B.size()) {
        return A.size() < B.size();                    // 位数不同，短的数一定更小
    }
    for (int i = (int)A.size() - 1; i >= 0; --i) {     // 从最高位往低位比
        if (A[i] != B[i]) return A[i] < B[i];          // 找到第一位不同的即可定胜负
    }
    return false;                                      // 完全相等时，A < B 为假
}
```

本页要点

less_vec(A, B) 本身也是个严格弱序比较器：当 $A == B$ 时返回 false！

模板代码 (2/2)：高精度减法主循环

```
if (less_vec(A, B)) { swap(A, B); cout << '-'; }    // 先判大小，保证大减小
int borrow = 0;
vector<int> C;
for (size_t i = 0; i < A.size(); ++i) {
    int d = A[i] - borrow - (i < B.size() ? B[i] : 0);
    if (d < 0) { d += 10; borrow = 1; }             // 不够减，借 1 当 10
    else borrow = 0;                                // 够减必须及时把 borrow 归零！
    C.push_back(d);
}
while (C.size() > 1 && C.back() == 0) C.pop_back(); // 删掉高位消出来的 0
```

本页要点

最容易漏写的一行就是 `else borrow = 0;`——漏写它，借过一次位之后后面每一轮都会多减 1！

高精度减法上机验证（3 组数据 + 反事实追问）

使用完整程序 `code/big_sub.cpp` 验证三组核心场景：

- 连续借位消零：输入 `1000` 和 `1` → 预期输出 `999`。
- 相等结果为零：输入 `123` 和 `123` → 预期输出 `0`（靠 `C.size() > 1` 留住 0）。
- 小减大出负号：输入 `12` 和 `100` → 预期输出 `-88`。

反事实追问

如果把代码里的 `else borrow = 0;` 删掉，输入 `121` 和 `2` 时会输出什么？
答：个位 $1-2$ 借位得 9（`borrow=1`）；十位 $2-1=1$ 明明够减，却没把 `borrow` 清零，于是百位继续被多减 1 变成 0，结果错输出 `19` 而非 `119`（**WA**）！

高精度减法易错对照

错误类型	常见错误写法	正确做法与原因
WA（连环多减）	只写 `if (d < 0)` 忘写 `else borrow = 0;`	够减一次就必须把 `borrow` 复位成 0
WA（比错大小）	`less_vec` 从 `i = 0` 往高位比	个位小不代表数小（如 $19 < 20$），必须从高位往低位比
WA（留前导零）	减完没做 `pop_back` 去前导零	`1000 - 999` 会错输出 `0001`
WA（出负位数）	没比大小直接拿小减大	必先用 `less_vec` 交换并先打印 `'-'`

本页要点

减法写完必测三组：121-2（检验 borrow 复位）、1000-999（去前导零）、123-123（结果为零）！

生活引入：从 23 * 4 到 23 * 45 的乘法规律

两种常见乘法需求

1. 高精 $\times$ 单精：比如算 2^{100} 或 $100!$，是一个大整数不断乘以 2 或 i（普通 `int`）。
2. 高精 $\times$ 高精：比如算两个 200 位数相乘，两个乘数都要放进倒序数组。

- 高精乘单精度时，把单精度 b 当成一个整体去乘 A 的每一位，再处理进位（注意进位可能不止 1，比如乘 99 时进位可能是两位数）。
- 高精乘高精度时，怎么知道 A 的第 i 位乘以 B 的第 j 位应该加到结果的哪一格？

本页要点

把每个数字看成“数字 $\times 10^{\text{下标}}$”，指数相加立刻告诉你乘积落在哪一格！

核心公式：为什么 A[i] * B[j] 必须加进 C[i + j]？

在倒序数组里，$A[i]$ 实际代表数值 $A[i] \times 10^i$，$B[j]$ 实际代表数值 $B[j] \times 10^j$。两者相乘得到：

$$\left(A[i] \times 10^i\right) \times \left(B[j] \times 10^j\right) = (A[i] \cdot B[j]) \times 10^{i+j}$$

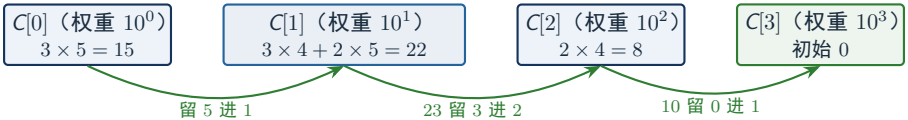
- 比如 23×45（倒序 $A = [3, 2],\ B = [5, 4]$）：
  - 个位 $A[0]$ 乘以十位 $B[1]$（$3 \times 40 = 120$）权位是 $10^{0+1} = 10^1$，落在 C[1]；
  - 十位 $A[1]$ 乘以个位 $B[0]$（$20 \times 5 = 100$）权位同样是 $10^{1+0} = 10^1$，也要累加进 C[1]！

本页要点

同一格子 $C[k]$ 会收到所有满足 $i + j = k$ 的乘积贡献，所以必须写 C[i + j] += A[i] * B[j]！

TikZ 图解：23 * 45 错位累加与统一进位

把 $A=[3,2]$（23）与 $B=[5,4]$（45）先全塞进 $C[i+j]$ 暂存，再从左往右统一整理进位：



- 统一进位后得到 $C=[5,3,0,1]$，倒着打印出来恰好就是 1035（$23\times45=1035$）！

本页要点

先全部累加、后统一进位，比在双层循环里边乘边进位简洁得多，也更不容易出错！

手算跟踪表：23 * 45 两阶段完整手算

阶段一（全部累加后）：$C = [15,\ 22,\ 8,\ 0,\ 0]$。阶段二（统一进位，$i = 0 \sim 4$）：

i	原 $C[i]$	$v = C[i] + carry$	写新 $C[i] = v \bmod 10$	新 $carry = v/10$	说明
0	15	$15 + 0 = 15$	5	1	个位留 5 进 1
1	22	$22 + 1 = 23$	3	2	十位留 3 进 2
2	8	$8 + 2 = 10$	0	1	百位留 0 进 1
3	0	$0 + 1 = 1$	1	0	千位接住进位 1
4	0	$0 + 0 = 0$	0	0	末尾多余 0 靠 `pop_back` 删掉

本页要点

最后清理前导零得到 `[5, 3, 0, 1]`，倒序打印为 `1035`！

正确性与空间上限：结果最多有多少位？会溢出 int 吗？

- 为什么结果数组开 `A.size() + B.size() + 1` 绝对够用？
  - m 位数乘以 n 位数，积最多只有 $m+n$ 位（例如最大的两位数相乘 $99 \times 99 = 9801$，刚好 $2+2=4$ 位）。开 $|A|+|B|+1$ 格不仅够放，还给末尾进位留足缓冲！
- 先累加再统一进位，中间格子会装爆 `int` 吗？
  - 若乘数长 200 位，一个格子最多累加 200 对 $9 \times 9 = 81$，最大值仅 $200 \times 81 = 16200$，远远小于 `int` 上限 2.14×10^9！

本页要点

只要乘数长度在百万以内，`vector<int>` 暂存累加和完全安全；如果乘单精度 $b \le 10^9$ 则要用 `long long`。

模板代码 (1/2)：高精度乘单精度

```
vector<int> mul_small(const vector<int> &A, long long b) {
    vector<int> C;
    long long carry = 0;                          // 防 A[i]*b 溢出
    for (size_t i = 0; i < A.size() || carry; ++i) {
        long long t = carry + (i < A.size() ? A[i] * b : 0);
        C.push_back((int)(t % 10));               // 留下当前个位
        carry = t / 10;                           // 多位进位继续推
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

本页要点

乘数为 0 时靠 `C.size() > 1` 保留一位 0；`|| carry` 自动拆开多位进位！

模板代码 (2/2)：高精度乘高精度

```cpp
vector<int> mul_big(const vector<int> &A, const vector<int> &B) {
    vector<int> C(A.size() + B.size() + 1, 0);  // 预留足够全 0 格
    for (size_t i = 0; i < A.size(); ++i)
        for (size_t j = 0; j < B.size(); ++j)
            C[i + j] += A[i] * B[j];             // 累加到权重位 i+j
    int carry = 0;
    for (size_t i = 0; i < C.size(); ++i) {
        int v = C[i] + carry;
        C[i] = v % 10; carry = v / 10;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

本页要点

先双重循环按 `i+j` 累加，再单层循环统一整理进位和去零！

高精度乘法上机验证（3 组数据 + 反事实追问）

使用完整程序 `code/mul_small_demo.cpp` 与 `code/big_mul.cpp` 测试三组边界：

- 小数据验证：输入 23 和 45 → 预期输出 1035。
- 乘零边界：输入 0 和 999999 → 预期输出 0（不能输出 0000000 或空串）。
- 末尾多零：输入 12345678900 和 98765432100 → 预期输出 1219326311126352690000。

反事实追问

如果写高精乘高精时把 `C[i + j] += A[i] * B[j];` 漏掉了加号写成 `=`，算 23 * 45 会变成多少？

答：`C[1]` 先算入 3*4=12，紧接着被 2*5=10 覆盖，丢掉 120，结果变成错误的 915（**WA**）！

高精度乘法易错对照

错误类型	常见错误写法	正确做法与原因
WA（覆盖累加）	写 `C[i+j] = A[i] * B[j]`	同一权位有多对乘积贡献，必须写 `+=`
RE（空间不足）	`C` 只开了 `max(lenA, lenB)` 格	积最多 $\|A\|+\|B\|$ 位，必须开 `A.size() + B.size() + 1`
WA（乘零留零）	乘以 0 忘做 `pop_back` 去前导零	必须写 `while (C.size() > 1 && C.back()==0)`
WA（高精乘单精）	乘完只用 `if (carry)` 进 1 位	单精度乘积进位可能大于 9，循环必带 `|| carry`

本页要点

高精乘高精“先错位累加、后统一进位”，高精乘单精“`long long` 防溢出 + `|| carry` 拆位”。

先停下来想：为什么除法不能倒序从个位算？

生活类比：把 268 颗糖平均分给 13 个小朋友

在小学列除法竖式时，你会先拿个位的 8 颗去分，还是先拿百位的 2 盒（200 颗）拆分？

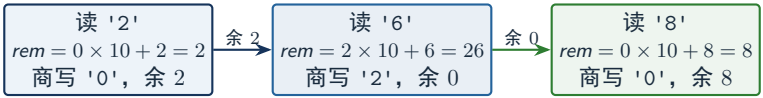
- 加、减、乘都是低位向高位进借，所以需要倒序从个位算；
- 而除法正好相反：先用最高位试除，不够除就把剩余的“余数 $\times 10$”加上下一位数字继续除！
- 因此高精度除以单精度时，根本不用倒序存数组，直接拿着原始正序字符串 `string s` 从左到右（从下标 0 到末尾）一扫到底！

本页要点

加减乘是“倒序从个位算”，高精度除以单精度是“正序从最高位算”！

TikZ 图解：268 / 13 正序逐位试商与余数传播

直接从左到右扫描正序字符串 "268"，每一步把上轮余数 rem 乘 10 拼上当前数字：



- 扫描结束得到商字符串 q = "020" 和最终余数 rem = 8。
- 再把 q 开头多余的 '0' 去掉，就得到最终商 "20"、余数 8！

本页要点

商字符串在最高位往往会产生前导 '0'（例如 "020"），输出前一定要清理！

手算跟踪表：268 / 13 全部步骤与去前导零

初始正序字符串 $s=$ "268"，$q=$ ""，$rem=0$，逐字符遍历：

字符	原 rem	rem*10+d	商位	新 rem	当前商串 q
'2'	0	$0\times10+2=2$	0	2	"0"
'6'	2	$2\times10+6=26$	2	0	"02"
'8'	0	$0\times10+8=8$	0	8	"020"
结束	去前导 '0'	首非零下标 $p=1$	-	余数 8	q.substr(1) = "20"

本页要点

注意商全为 0 的特殊情况（比如 $5\div13$）：q = "0" 找不到非 '0'，必须特判直接输出 "0"！

正确性证明：正序除法的余数不变量

不变量阶段	要验证的内容
初始化	处理 0 个字符前，已得到的商 $Q=0$，余数 $rem=0$，满足 $0\times b+0=0$
保持	前缀数扩充一位变成 $10\cdot P+d$；由 $P=Q\cdot b+rem$ 可得 $10\cdot P+d=(10Q)\cdot b+(10\cdot rem+d)$，再把 $10\cdot rem+d$ 除以 b 得商位与新余数，等式始终严格成立！
终止	读完最后一位后，商串数值 Q 与最终余数 $rem\in[0,b-1]$ 严格满足 $N=Q\cdot b+rem$

本页要点

每一轮本质上都在维护等式 $N_{\text{前}i\text{ 位}}=Q_{\text{前}i\text{ 位}}\times b+rem$！

高精度除以单精度核心模板代码

```
string s, q;
long long b, rem = 0;                                   // 除数可达 1e9 时 rem 必须
    long long
cin >> s >> b;
for (char ch : s) {
    rem = rem * 10 + (ch - '0');                        // 上一位余数乘 10 加上当前位
    q.push_back(char('0' + rem / b));                   // 写出当前这一位商
    rem %= b;                                           // 留下新余数给下一位
}
size_t p = q.find_first_not_of('0');                    // 找第一个不是 '0' 的位置
cout << (p == string::npos ? "0" : q.substr(p)) << '\n' << rem << '\n';
```

本页要点

用 `q.find_first_not_of('0')` 配 `string::npos` 判全零，一行解决前导 `'0'` 和商为 0 两个坑！

高精度除法上机验证（3 组数据 + 反事实追问）

使用完整程序 `code/div_13.cpp`（除数为 13）测试三组边界用例：

- 普通整数：输入 268 → 预期输出商 20、余数 8。
- 被除数小于除数：输入 5 → 预期输出商 0、余数 5。
- 大整数整除：输入 2132104848488485 → 预期输出商 164008065268345、余数 0。

反事实追问

如果输入 5（除以 13），把输出商那一行简化成 `cout << q.substr(q.find_first_not_of('0'));` 而不判断 `string::npos` 会发生什么？
答：`q = "0"` 全是 `'0'`，找不到非零字符会返回 `npos`（极大无符号值），调用 `q.substr(npos)` 直接抛出越界异常崩溃 **RE**！

高精度除法易错对照

错误类型	常见错误写法	正确做法与原因
WA（顺序用反）	把被除数倒序后再从下标 0 除	除法必须从正序最高位开始向低位除
WA（余数溢出）	除数可达 10^9 时 `rem` 开成 `int`	`rem * 10` 最高可达 10^{10}，必开 `long long`
RE（商零崩溃）	未判断 `p == string::npos` 直接 `substr`	找不到非 `'0'` 说明商是 0,特判输出 `"0"`
WA（漏加编码）	往商串里写 `q.push_back(rem / b)`	忘加 `'0'` 会输出不可见控制字符，必写 `'0' + rem / b`

本页要点

记住“正序除、`long long rem`、商位加 `'0'`、特判 `npos` 输出 `0`”四句话！

[入门] 例 1 | 大整数加法（一本通 1168）：先读懂要求

大白话题意

计算两个不超过 200 位的非负整数 a 与 b 的和。注意题目明确说明：输入的整数可能带有多余的前导 0，输出的结果绝不能含有多余前导 0。

1. 输入：两行，每行一个不超过 200 位的非负整数（可能带前导 0）。
2. 处理：倒序存入数组，逐位累加并进位，最后清除最高位多余 0。
3. 输出：一行，$a+b$ 的精确结果。

原题样例

输入 2222222222222222222 与 33333333333333333333
$\Rightarrow$ 输出 55555555555555555555。

[入门] 例 1 | 建模技巧与样例手推

- 怎么处理输入自带的前导 0?
  - 不需要读入时写复杂字符串截取！倒序存进数组后，输入前面多余的 0 全都被甩到了数组末尾。
  - 加法算完后运行一遍 `while (C.size() > 1 && C.back() == 0) C.pop_back();`，就把输入前导 0 和结果为 0 的边界一次全搞定了！

手推带前导零小例子 `"00123" + "00077"`（倒序 $A = [3,2,1,0,0]$, $B = [7,7,0,0,0]$）：

按顺序计算	当前 C	进位	核对问题
加第 $0 \sim 2$ 位	`[0, 0, 2]`	0	个十百位刚好算出 $123 + 77 = 200$
加第 $3 \sim 4$ 位	`[0, 0, 2, 0, 0]`	0	输入前导零加完留在末尾
去前导零后	`[0, 0, 2]`	-	倒序打印 **200**，完全正确！

[入门] 例 1 | 完整可编译代码 (1/2)（code/big_add.cpp）

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

// 一本通 1168 大整数加法：倒序存储，支持输入含前导零
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string a, b;
    if (!(cin >> a >> b)) return 0;
    vector<int> A, B, C;
    for (int i = (int)a.size() - 1; i >= 0; --i) A.push_back(a[i] - '0');
    for (int i = (int)b.size() - 1; i >= 0; --i) B.push_back(b[i] - '0');
```

本页要点

直接把两串字符倒序塞进 A 和 B，哪怕各自前导零个数不一样也能个位对齐。

[入门] 例 1 | 完整可编译代码 (2/2)（code/big_add.cpp）

```
    int carry = 0;
    for (size_t i = 0; i < A.size() || i < B.size() || carry; ++i) {
        int s = carry + (i < A.size() ? A[i] : 0) + (i < B.size() ? B[i] : 0);
        C.push_back(s % 10);                          // 留下个位
        carry = s / 10;                               // 产生向高位的进位
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back(); // 清除前导 0
    for (int i = (int)C.size() - 1; i >= 0; --i) cout << C[i];
    cout << '\n';
    return 0;
}
```

本页要点

全部代码精简实用，一套 pop_back 扫尾处理掉前导零与全零边界。

[入门] 例 1 | 易错对照与三组自测边界

常见错误对照

- **WA**：没有清理前导 0，导致 `001 + 002` 输出 `003`；
- **WA**：只写 `!C.empty()` 去零，导致 `000 + 000` 输出空行；
- **WA**：循环漏写 `|| carry`，导致 `999 + 1` 丢掉最高位。

三组必测自检数据

1. 全零加全零：`000 + 000` $\rightarrow$ `0`；
2. 带前导零：`00123 + 00077` $\rightarrow$ `200`；
3. 连续进位：`9999999999999999999 + 1` $\rightarrow$ `10000000000000000000`。

本页要点

这 3 组自测全过，高精加法模板就彻底稳了！

[基础] 例 2 | 大整数减法（一本通 1169）：先读懂要求

大白话题意

求两个不超过 200 位的正整数 a 与 b 的差 $a-b$。原题保证 $a>b$ 且输入不含多余前导 0，要求输出结果也不能含有多余前导 0。

1. 输入：两行，被减数 a 与减数 b（各不超过 200 位）。
2. 处理：倒序存位并逐位相减借位；为通用起见，建议同时保留 `less_vec` 判负保护。
3. 输出：一行，$a-b$ 的差（不含多余前导 0）。

原题样例

输入 9999999999999999999999999999999999999 与 999999999999
⇒ 输出 9999999999999999999999999000000000000。

[基础] 例 2 | 建模分析与核心步骤表

虽然原题保证 $a > b$，但在正式模板里加上 `less_vec(A, B)` 只需 5 行，却能让代码通吃 $a < b$ 和 $a == b$ 所有变形题！

按顺序计算步骤	纸上得到（以 `1000 - 999` 为例）	核对问题
1. 倒序读入并判大小	$A = [0,0,0,1]$，$B = [9,9,9]$，$A > B$	若 $A < B$ 则交换并打 `'-'`
2. 逐位相减与借位	前三位借 1 得 0，第 3 位 $1 - 1 = 0$	够减时是否写 `borrow = 0`？
3. 清理最高位前导零	`[0, 0, 0, 0]` 弹出 3 个 `0`？	错！$1000 - 999$ 是 `[1,0,0,0]` 去零得 `1`

本页要点

做减法最容易产生大串高位前导 0（例如 $1000 - 999$ 消掉三位），收尾 `pop_back` 绝对不能省！

[基础] 例 2 | 完整可编译代码 (1/2)（code/big_sub.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;
bool less_vec(const vector<int> &A, const vector<int> &B) {
    if (A.size() != B.size()) return A.size() < B.size();
    for (int i = (int)A.size() - 1; i >= 0; --i)
        if (A[i] != B[i]) return A[i] < B[i];
    return false;
}
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    string a, b; if (!(cin >> a >> b)) return 0;
    vector<int> A, B, C;
    for (int i = (int)a.size() - 1; i >= 0; --i) A.push_back(a[i] - '0');
    for (int i = (int)b.size() - 1; i >= 0; --i) B.push_back(b[i] - '0');
```

注：先写好 less_vec 并倒序读入，确保接下来能放心算“大减小”。

[基础] 例 2 | 完整可编译代码 (2/2)（code/big_sub.cpp）

```
    if (less_vec(A, B)) { swap(A, B); cout << '-'; } // 保证大减小
    int borrow = 0;
    for (size_t i = 0; i < A.size(); ++i) {
        int d = A[i] - borrow - (i < B.size() ? B[i] : 0);
        if (d < 0) { d += 10; borrow = 1; }          // 向高位借 10
        else borrow = 0;                             // 够减立刻复位
        C.push_back(d);
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    for (int i = (int)C.size() - 1; i >= 0; --i) cout << C[i];
    cout << '\n'; return 0;
}
```

本页要点

循环里必须同时写清 `d < 0` 借位和 `else borrow = 0` 复位两条分支！

[基础] 例 2 | 易错对照与三组自测边界

常见错误对照

- **WA**：漏写 `else borrow = 0`，借过一次之后后面全错；
- **WA**：结果高位连续消掉多位却忘记去前导 0；
- **WA**：误把 `less_vec` 写成从下标 0 比到末尾。

三组必测自检数据

1. 连续借位且大消零：`1000 - 999` $\rightarrow$ `1`；
2. 两数相等：`555 - 555` $\rightarrow$ `0`；
3. 拓展负数测试：`9 - 100` $\rightarrow$ `-91`。

本页要点

用 `1000 - 999` 测试能同时检验“连续借位”和“消除三位前导零”两个考点！

[基础] 例 3 | 大整数乘法（一本通 1174）：先读懂要求

大白话题意

求两个不超过 200 位的非负整数 a 与 b 的乘积，输入不含多余前导 0，输出同样不能含多余前导 0。

1. 输入：两行，两个不超过 200 位的非负整数。
2. 处理：倒序存位，用双层循环把 $A[i] \cdot B[j]$ 累加到 $C[i+j]$，再统一进位去零。
3. 输出：一行，$a \times b$ 的精确乘积。

原题样例

输入 12345678900 与 98765432100
$\Rightarrow$ 输出 1219326311126352690000。

[基础] 例 3 | 建模手推与核心步骤表

观察样例结尾各带 2 个 0，倒序后 $A[0]=A[1]=0$，$B[0]=B[1]=0$，于是 $C[0..3]$ 自然全为 0，结果末尾刚好 4 个 0！

按顺序计算步骤	纸上得到（以 `120 * 30` 为例）	核对问题
1. 开结果数组	长度开 `3 + 2 + 1 = 6` 格全 0	保证绝对够放不越界
2. 错位累加	$A=[0,2,1], B=[0,3] \rightarrow C=[0,0,6,3,0,0]$	写 `+=` 而非 `=`
3. 统一处理进位	每位对 10 取余并把商进到下一位	如果 $C[i] \geq 10$ 会向后推进
4. 清除前导零	`pop_back` 弹出末尾两个 `0`	若有乘数为 `0`，仍留一位 `0`

本页要点

千万不要用手算跳过末尾 0 的小聪明，倒序错位累加本来就能自动把末尾 0 对齐好！

[基础] 例 3 | 完整可编译代码 (1/2)（code/big_mul.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    string a, b; if (!(cin >> a >> b)) return 0;
    vector<int> A, B;
    for (int i = (int)a.size() - 1; i >= 0; --i) A.push_back(a[i] - '0');
    for (int i = (int)b.size() - 1; i >= 0; --i) B.push_back(b[i] - '0');
    vector<int> C(A.size() + B.size() + 1, 0);
    for (size_t i = 0; i < A.size(); ++i)
        for (size_t j = 0; j < B.size(); ++j) C[i + j] += A[i] * B[j];
```

本页要点

先开全 0 数组 C(A.size() + B.size() + 1, 0)，再把局部乘积用 += 累加。

[基础] 例 3 | 完整可编译代码 (2/2)（code/big_mul.cpp）

```
    int carry = 0;
    for (size_t i = 0; i < C.size(); ++i) {
        int v = C[i] + carry;
        C[i] = v % 10;                          // 统一整理当前位
        carry = v / 10;                         // 向下一位传递进位
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    for (int i = (int)C.size() - 1; i >= 0; --i) cout << C[i];
    cout << '\n';
    return 0;
}
```

本页要点

两个 200 位数相乘双重循环只做 $200 \times 200 = 40000$ 次乘法，不到 1 毫秒即可出结果！

[基础] 例 3 | 易错对照与三组自测边界

常见错误对照

- **WA**：错位累加漏写加号变成 `C[i+j] = A[i]*B[j];`
- **WA**：乘以 0 时没有去前导零，输出一长串 `00000`；
- **RE**：数组长度只开到 205 而非 405，两位 200 位数相乘越界。

三组必测自检数据

1. 乘以零：`0 * 98765432100` $\rightarrow$ `0`；
2. 最高位连续进位：`999 * 999` $\rightarrow$ `998001`；
3. 原题样例：`12345678900 * 98765432100` $\rightarrow$ `1219326311126352690000`。

本页要点

遇到乘数是 0 时，是检验去前导零有没有保留至少 1 位的关键测试！

[基础] 例 4 | 除以 13（一本通 1175）：先读懂要求与手推

大白话题意与原题样例

输入一个位数不超过 100 的正整数 N，两行分别输出 $N \div 13$ 的商 q 和余数 r（商为 0 时输出 0）。

样例输入：2132104848488485 $\Rightarrow$ 样例输出：164008065268345（第 1 行）和 0（第 2 行）。

按顺序计算步骤	纸上得到（以样例前 4 位 2132 为例）	核对问题
1. 读 '2'	$rem = 2$，商 '0'，新 $rem = 2$	首字符商 '0' 记下来，待删
2. 读 '1'	$rem = 21$，商 '1'，新 $rem = 8$	$21 = 1 \times 13 + 8$
3. 读 '3'	$rem = 83$，商 '6'，新 $rem = 5$	$83 = 6 \times 13 + 5$
4. 读 '2'	$rem = 52$，商 '4'，新 $rem = 0$	去掉前导 '0' 得 164...

[基础] 例 4 | 完整可编译代码 (1/2)（code/div_13.cpp）

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

// 一本通 1175 除以 13：高精度除以低精度，从最高位正序推进
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string s, q;
    if (!(cin >> s)) return 0;
```

本页要点

正序字符串 s 读入后直接遍历即可，完全不需要转换成倒序数组。

[基础] 例 4 | 完整可编译代码 (2/2)（code/div_13.cpp）

```
    int rem = 0;
    for (char ch : s) {
        rem = rem * 10 + (ch - '0');                // 上一位余数乘 10 加上当前位
        q.push_back(char('0' + rem / 13));          // 产生这一位商
        rem %= 13;                                  // 更新余数给下一位使用
    }
    size_t p = q.find_first_not_of('0');            // 去掉商前面的多余前导 0
    cout << (p == string::npos ? "0" : q.substr(p)) << '\n' << rem << '\n';
    return 0;
}
```

本页要点

若某道题除数可达 10^9，记得把第 1 行的 `int rem = 0;` 改成 `long long rem = 0;`！

[基础] 例 4 | 易错对照与三组自测边界

常见错误对照

- **RE**：输入小于 13 的数时没判 `p == string::npos` 直接 `substr`；
- **WA**：习惯性把输入倒序后再除，结果彻底算错；
- **PE / WA**：商和余数中间输出空格而非换行（题目要求两行）。

三组必测自检数据

1. 商为零：12 → 商 0，余数 12；
2. 刚好整除：13 → 商 1，余数 0；
3. 商中间有零：268 → 商 20，余数 8（中间的 `'0'` 不能误删）。

本页要点

`find_first_not_of('0')` 只删掉左端前导 `'0'`，商中间和末尾的 `'0'` 会被安全保留！

[提高] 例 5 | 回文数（NOIP 1999）：先读懂要求

大白话题意

如果一个数从左往右读和从右往左读一模一样，就叫回文数。给定进制 N（$2 < N \le 10$ 或 $N = 16$）和该进制下的一个数 M，每步把 M 加上它的逆序数（做一次 N 进制加法）。问最少几步能得到回文数？如果在 30 步以内（含第 30 步）无法得到，输出 `Impossible`。

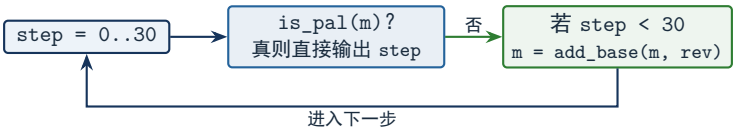
1. 输入：第 1 行进制 N；第 2 行 N 进制数 M（16 进制用 `A~F` 表示 $10 \sim 15$）。
2. 处理：从 $step = 0$ 开始检查是否回文，不是就做一次 N 进制加法，最多做 30 步。
3. 输出：一行最少步数或 `Impossible`。

原题样例

输入 9 和 87 $\Rightarrow$ 输出 6。

[提高] 例 5 | 怎么想到的与 TikZ 建模图

- 每步相加一次，位数最多增 1，最多 30 步会远远超过 `long long`（尤其原始位数已较长时），必须用高精度加法！
- 怎么把十进制高精加变成 N 进制？只需把 `% 10`，`/ 10` 全改成 `% base`，`/ base`！
- 十六进制字符转数字：`'0'~'9'` 转 `ch - '0'`，`'A'~'F'` 转 `toupper(ch) - 'A' + 10`。



本页要点

先判回文、再决定加不加！这样本身就是回文（$step = 0$）和第 30 步恰好变回文都能算对！

[提高] 例 5 | 样例手推表（9 进制 87 走完 6 步）

十进制加法满 10 进 1，9 进制加法则是满 9 进 1（逢 9 减 9 进 1）：

步数 step	当前数 M（9 进制）	逆序数	9 进制相加结果	是否回文？
0	87_9	78_9	$87_9 + 78_9 = 176_9$（$7+8=15=1\cdot 9+6$）	否
1	176_9	671_9	$176_9 + 671_9 = 857_9$	否
2	857_9	758_9	$857_9 + 758_9 = 1726_9$	否
3	1726_9	6271_9	$1726_9 + 6271_9 = 8107_9$	否
4	8107_9	7018_9	$8107_9 + 7018_9 = 16126_9$	否
5	16126_9	62161_9	$16126_9 + 62161_9 = 78287_9$	下轮判真
6	78287_9	-	满足回文条件	输出 6！

本页要点

第 0 步先检查 87_9 不是回文再做第 1 次加法；共加了 6 次，到 `step = 6` 判出回文停下！

[提高] 例 5 | 完整可编译代码 (1/3)（code/palindrome.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;
int base;
vector<int> add_base(const vector<int> &a, const vector<int> &b) {
    vector<int> c; int carry = 0;
    for (size_t i = 0; i < a.size() || i < b.size() || carry; ++i) {
        int s = carry + (i < a.size() ? a[i] : 0)
                      + (i < b.size() ? b[i] : 0);
        c.push_back(s % base);                  // 按 base 进制取模进位
        carry = s / base;
    }
    return c;
}
```

本页要点

add_base 与十进制高精度加法结构相同，只把 10 换成了 base！

[提高] 例 5 | 完整可编译代码 (2/3)（code/palindrome.cpp）

```cpp
bool is_pal(const vector<int> &a) {
    for (size_t i = 0, j = a.size() - 1; i < j; ++i, --j)
        if (a[i] != a[j]) return false;
    return true;
}
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    string mstr; if (!(cin >> base >> mstr)) return 0;
    vector<int> m;
    for (int i = (int)mstr.size() - 1; i >= 0; --i) {
        int c = (unsigned char)mstr[i];
        m.push_back(isdigit(c) ? c - '0' : toupper(c) - 'A' + 10);
    }
```

注：用 `toupper` 兼容大小写十六进制字母（`'a'` 与 `'A'` 都映射成 10）。

[提高] 例 5 | 完整可编译代码 (3/3)（code/palindrome.cpp）

```
    while (m.size() > 1 && m.back() == 0) m.pop_back();
    for (int step = 0; step <= 30; ++step) {
        if (is_pal(m)) { cout << step << '\n'; return 0; }
        if (step == 30) break;                          // 第 30 步仍非回文直接退出
        vector<int> rev(m.rbegin(), m.rend());
        m = add_base(m, rev);
    }
    cout << "Impossible\n";
    return 0;
}
```

本页要点

循环跑 `step = 0..30`：第 30 次判完还不是回文就 `break`，避免多加出第 31 步！

[提高] 例 5 | 易错对照与三组自测边界

常见错误对照

- **WA**：第 0 步没先判回文，导致本来就是回文输出 1；
- **WA**：循环写成 `step < 30`，漏掉恰好在第 30 步变回文的情况；
- **WA**：只处理了十进制 `'0'~'9'`，遇到 16 进制 `A~F` 出错。

三组必测自检数据

1. 0 步回文：10 和 121 → 输出 0；
2. 十六进制：16 和 A7 → $A7_{16} + 7A_{16} = 121_{16}$，输出 1；
3. 超过 30 步无解：10 和 196 → 输出 `Impossible`。

本页要点

掌握进制通用高精度加法后，任何 K 进制大整数加法都能一分钟改出来！

高精度运算常见错误对照与定位指南

运算 / 场景	常见错误写法	正确做法与原因
加法漏进位	条件写 `i < max(|A|, |B|)`	必带 `|| carry`，否则 999+1 丢最高位 1
减法借位传播	只写 `if(d<0)` 忘 `else borrow=0`	借完一位后后续全被多减 1（**WA**）
减法小减大	没判大小直接拿小数减大数	先用 `less_vec` 交换并先打 `'-'` 算大减小
乘法写覆盖	写 `C[i+j] = A[i] * B[j]`	同一权位有多对乘积贡献，必须写 `+=`
乘法与去零	写 `while (!C.empty() && C.back()==0)`	算成 0 会删光变空输出，必写 `C.size() > 1`

本页要点

定位口诀：输出空行查 `C.size()>1`；少最高位查 `|| carry`；高位整体少 1 查 `borrow=0`！

提交前检查清单（六必查）

1. 是否用 `string` 读入大整数，并在加减乘时从字符串末尾往前倒序存进数组？
2. 高精加循环是否带有 `|| carry`？高精乘高精数组长度是否开到了 `A.size() + B.size() + 1`？
3. 高精减是否先判过 $A < B$ 的负号交换，并在每次够减时把 `borrow` 复位成 0？
4. 去前导零循环是否写的是 `while (C.size() > 1 && C.back() == 0) C.pop_back();`？
5. 高精除以单精度是否从正序字符串的第 0 字符（最高位）推进，且特判了商全为 0（`npos`）？
6. 倒序输出时下标变量是否用的有符号 `int`（而不是会下溢死循环的 `size_t`）？

最后一步

每次高精度写完，必测 `0 op 0`、`999 + 1`、`1000 - 999` 这“三剑客”用例再提交！

本章全部练习与完成标准

来源	题名	难度	核心考点与对应源码
1168	大整数加法	[入门]	倒序加法 + 处理输入前导零（big_add.cpp）
1169	大整数减法	[基础]	借位复位 + 清理消位前导零（big_sub.cpp）
1170	计算 2 的 N 次方	[入门]	高精乘单精度 $b = 2$ 循环 N 次（mul_small_demo.cpp）
1174	大整数乘法	[基础]	C[i+j] += A[i]*B[j] 错位累加（big_mul.cpp）
1175	除以 13	[基础]	正序长除法求商与余数（div_13.cpp）
1173	阶乘和	[基础]	高精乘单精（算 $i!$）与高精加结合累加
1309	回文数（NOIP 99）	[提高]	N 进制高精度加法与 30 步回文判定（palindrome.cpp）

本页要点

完成标准：能不看模板手写高精度加减乘除核心循环，并独立 AC 全部 7 题。

课后练习路线（从入门到提高）

1. 第一阶段（筑牢加减与单精乘）：先 AC《大整数加法》（1168）、《大整数减法》（1169）和《计算 2^N》（1170），把“倒序对齐 + 去前导零”写成本能。
2. 第二阶段（打通乘除与组合）：独立写出《大整数乘法》（1174）、《除以 13》（1175）以及把高精乘和高精加组合起来的《阶乘和》（1173）。
3. 第三阶段（进制拓展挑战）：挑战《回文数》（1309），练习把十进制竖式推广到任意 N 进制。

复习动作

合上课件，在草稿纸上画一遍 23×45 错位累加图，口述为什么 `A[i] * B[j]` 必须加到 `C[i + j]` 里。

课后安排与下一章预告

- 源码对照：把 `code/` 下全部源码重打编译并测试 0、进借位跨多位以及超长位数数据。
- 下一章预告：第 4 章《二分答案与三分法》——前两章我们学会了把数据排好序和用数组精确算大数；下一章我们要学一种考试最常见的思维大反转：当直接算最优解太难时，如何用“先猜一个答案，再在有序范围内 $O(n)$ 检验它行不行”的二分答案思想一举破题！

本章一句话

加减乘倒序对齐按位进借，除法正序顺着余数往下推，收尾永远留住最后一位 0。

谢谢！