第一章

高精度计算

把数字存进数组，把竖式写进代码

HGNU-ACM 实验室算法组

第一章　配套：4 道例题，9 道练习题





本章目录 | 上半

为什么需要高精度

数字怎么存

竖式加法

竖式减法

竖式乘法

本章目录 | 下半

高精除

回文数与进制

易错点、模板与扩展

本章练习索引

内置类型到底能存多大的数

类型	位宽	大致能表示的十进制位数
`int`	32	约 10 位（最大 2147483647）
`long long`	64	约 19 位（最大 9223372036854775807）
`__int128`	128	约 38 位

- $2^{100} = 1267650600228229401496703205376$，有 31 位。
- $100!$ 有 158 位；$1000!$ 有 2568 位；$10000!$ 有 35660 位。
- 位数每多一位，可表示的范围就扩大 10 倍，增长非常快。

结论

输入或答案一旦超过 18 位，内置整数类型就装不下，必须把数字按位存起来自己算。

为什么 int 会溢出：8 位玩具模型

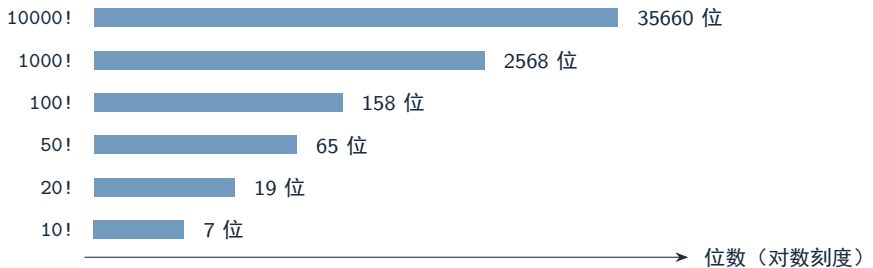


- 8 位只能表示 256 个不同的值；加到 127 再加 1 就“绕回”到 -128。
- 32 位的 int 同理：$2147483647 + 1$ 会得到 -2147483648。
- C++ 中有符号整数溢出是未定义行为，不能依赖这种回绕结果。

根本原因

位数是固定的，最高位的进位没有地方存放。

long long 也不够：阶乘位数增长有多快



- 20! 有 19 位，刚好卡在 long long 的边缘。
- 100! 有 158 位，10000! 有 35660 位，远超任何内置类型。

判断

只要题目给出的上限超过 19 位，或者运算结果会超过 19 位，就必须使用高精度。

什么时候必须手写高精度

- 输入的整数长度超过 18 位，例如“不超过 200 位”“小于 200000 位”。
- 运算结果增长很快：阶乘、乘方、累加、斐波那契数列的较大项。
- 题目要求输出精确值，不能用浮点数近似。
- 如果题目只问“对 p 取模的结果”，通常不需要高精度，直接边算边取模。

不要用 `double` 代替

`double` 只有约 $15 \sim 16$ 位有效数字；$100!$ 的低位会被全部舍去，答案必然错误。

- 高精度的本质：把一个大整数拆成很多个 $0 \sim 9$ 的数字，逐位模拟手算。

本章路线：先存对，再算对

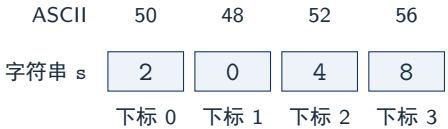
1. 存储：字符串读入，倒序放进数组，处理好前导零。
2. 加法：从个位起逐位相加，进位参与下一位。
3. 减法：不够减就向高位借 1 当 10。
4. 乘法：逐位相乘，按 $i+j$ 对位累加，最后统一进位。
5. 除法：从最高位起逐位落位、试商、求余数。

四道例题

例 1.2 高精度加法；例 1.3 高精度乘法；例 1.5 高精除；例 1.6 回文数。

学习方法：每个模板先在纸上推一遍样例，再写代码；最后用小数据和边界数据自查。

字符串：数字进入程序的第一站

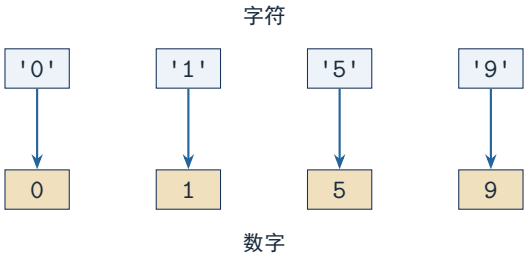


- `string` 本质上是一个字符数组，每个字符占 1 字节。
- 字符 `'0'` 到 `'9'` 的 ASCII 码是 48 到 57，并且连续排列。
- `s.size()` 返回无符号的 `size_t`；与 `int` 比较或相减时要先转换。

读入方式

用 `string` 读入，可以容纳几十万位，不会像 `int` 那样溢出。

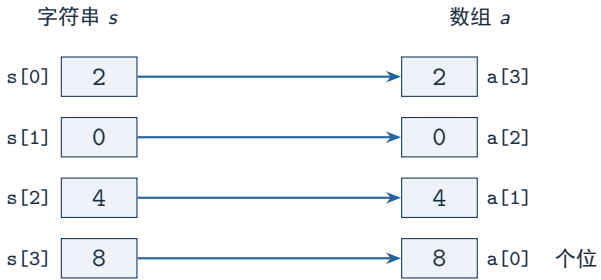
字符与数字：一步转换



$$d = \texttt{ch} - \texttt{'0'}, \qquad \texttt{ch} = d + \texttt{'0'}.$$

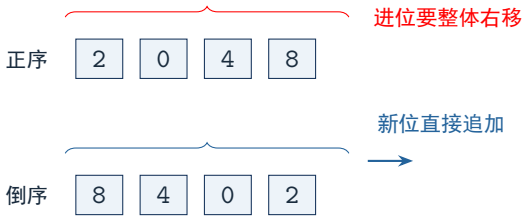
- 不要写死 48、57 这样的“魔法数”，用 `'0'` 更清楚。
- 只对数字字符有效；读入十六进制时，还要处理 `A`～`F`。
- 输出时可以直接 `cout << char(d + '0')`，不必查 ASCII 表。

倒序存储：a[0] 是个位



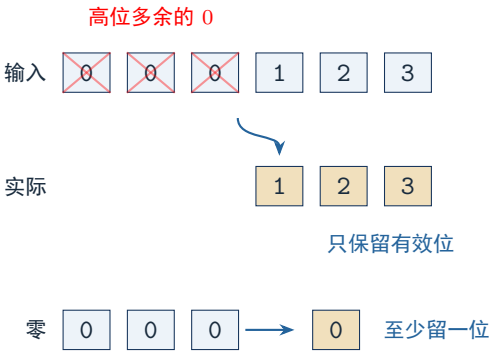
- 读入 s 后，从 s.size()-1 往 0 扫描，依次 push_back。
- a[0] 是个位，a[1] 是十位，$\cdots$，a[i] 表示 10^i 这一位。
- 数组的长度就是当前数的位数；数组下标就是“10 的幂次”。

为什么倒序：对齐、进位、变长



- 对齐：个位与个位都在下标 0，十位与十位都在下标 1。
- 进位：进位从低位走向高位，正好是从小下标走向大下标。
- 变长：结果多出一位时，直接 `push_back` 到末尾。
- 若正着存，每次进位都要在数组头部插入，既麻烦又慢。

前导零：从哪来，为什么要去



- 输入可能有前导零："000123" 与 "123" 是同一个数。
- 减法、乘法也可能在结果高位留下 0，输出前必须去掉。
- 数字 0 统一存成 $\{0\}$，不要存成空数组。

去前导零的代码与边界

```
void trim(vector<int>& a) {
    while (a.size() > 1 && a.back() == 0) {
        a.pop_back(); // 只删最高位的 0
    }
    if (a.empty()) {
        a.push_back(0); // 至少保留一个 0
    }
}
```

- 倒序存储时，back() 就是最高位，所以从末尾删。
- 条件 size() > 1 防止把唯一的 0 也删掉。
- 空数组是非法状态；加上空数组判断，程序更稳。

读入模板：字符串 → 倒序数组

```
string s;
cin >> s; // 例: s = "00123"
vector<int> a;
for (int i = (int)s.size() - 1; i >= 0; --i) {
    a.push_back(s[i] - '0'); // 从个位开始存
}
while (a.size() > 1 && a.back() == 0) {
    a.pop_back(); // 去掉高位前导零
}
```

- 输入保证是非空字符串；若可能为空，要先特判。
- 先全部转成数字，再去前导零，顺序不能反。
- `(int)s.size()` 把无符号数转成有符号，避免减到 -1 时下溢。

输出模板：倒序数组 → 屏幕




```
for (int i = (int)a.size() - 1; i >= 0; --i) {
    cout << a[i]; // 先输出最高位
}
cout << '\n';
```

- 输出方向与存储方向相反：从末尾的下标一直走到 0。
- 如果结果数组为空，先补一个 0 再输出。

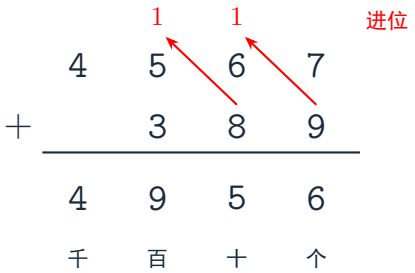
存储阶段自测

1. 字符串 "000123" 倒序存进数组后，数组内容是什么？
2. 字符串 "0" 存进数组后，数组内容是什么？
3. 数组 {0,0,1} 表示哪个数？直接倒序输出会得到什么？
4. 为什么不能把数字正着存进数组？
5. 去前导零时，为什么循环条件必须写 `size() > 1`？

参考答案

(1) {3,2,1,0,0,0}；(2) {0}；(3) 表示 100，输出 001，必须先删掉高位的两个 0；(4) 进位要在数组头部插入，效率低且容易错；(5) 防止把唯一的 0 删成空数组。

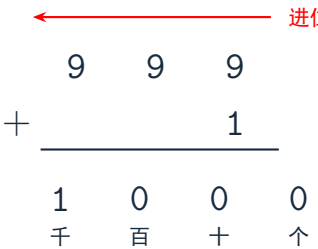
竖式加法：从个位开始，逐位相加



- 适用条件：两个非负整数，位数可以很大，按十进制逐位处理。
- 从个位（下标 0）开始，逐位相加，进位参与下一位。
- 每一位的和最多 $9+9+1=19$，所以进位只能是 0 或 1。

$$c_i = (a_i + b_i + carry) \bmod 10, \qquad carry = \left\lfloor \frac{a_i + b_i + carry}{10} \right\rfloor$$

进位链：$999 + 1$ 为什么会变成 1000



- 个位 $9 + 1 = 10$，写 0 进 1；十位 $9 + 0 + 1 = 10$，继续进位。
- 连续进位会一直传到最高位，最后还要在最高位前面补一个 1。
- 循环条件里的 `|| carry` 就是为这一步准备的。

结果长度

和最多有 $\max(n, m) + 1$ 位；数组要按这个长度准备，或在末尾直接追加。

加法模板：循环、取余、进位

```cpp
int carry = 0; // 上一位的进位
int n = max(A.size(), B.size());
vector<int> C;
for (int i = 0; i < n || carry; ++i) {
    int sum = carry;
    if (i < (int)A.size()) sum += A[i];
    if (i < (int)B.size()) sum += B[i];
    C.push_back(sum % 10); // 本位
    carry = sum / 10; // 新进位
}
```

- 循环条件 `i < n || carry`：即使两个数都加完了，最后的进位仍要处理。
- 时间 $O(\max(n, m))$，空间 $O(\max(n, m))$。
- 最后别忘了去掉结果高位的 0。

加法函数：封装成 addBig

```
vector<int> addBig(const vector<int>& A, const vector<int>& B) {
    vector<int> C;
    int carry = 0;
    int n = max(A.size(), B.size());
    for (int i = 0; i < n || carry; ++i) {
        int sum = carry;
        if (i < (int)A.size()) sum += A[i];
        if (i < (int)B.size()) sum += B[i];
        C.push_back(sum % 10);
        carry = sum / 10;
    }
    while (C.size() > 1 && C.back() == 0) {
        C.pop_back();
    }
    return C;
}
```

例 1.2 | 高精度加法：题意与样例手推

题意

输入两个不超过 100 位的非负整数，输出它们的和。

样例是两个 20 位数：

$$\begin{array}{r} 22222222222222222222 \\ +\,33333333333333333333 \\ \hline 55555555555555555555 \end{array}$$

- 每一位都是 $2+3=5$，没有进位，所以答案仍是 20 位。
- 样例太“整齐”，必须自己补测：$999+1=1000$ 会发生连续进位。
- 思路：倒序存储 $\rightarrow$ 逐位相加 $\rightarrow$ 处理进位 $\rightarrow$ 去前导零 $\rightarrow$ 倒序输出。

例 1.2 | 完整程序（1/2）：读入与倒序

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

int main() {
    string sa, sb;
    cin >> sa >> sb; // 读入两个字符串
    vector<int> A, B;
    for (int i = (int)sa.size() - 1; i >= 0; --i) {
        A.push_back(sa[i] - '0');
    }
    for (int i = (int)sb.size() - 1; i >= 0; --i) {
        B.push_back(sb[i] - '0');
    }
    vector<int> C;
    int carry = 0;
```

下一页接主循环；两个数的个位都放在下标 0。

例 1.2 | 完整程序（2/2）：相加与输出

```
    int n = max(A.size(), B.size());
    for (int i = 0; i < n || carry; ++i) {
        int sum = carry;
        if (i < (int)A.size()) sum += A[i];
        if (i < (int)B.size()) sum += B[i];
        C.push_back(sum % 10);
        carry = sum / 10;
    }
    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;
}
```

例 1.2 | 复杂度与易错点

- 时间 $O(L)$，$L = \max(|a|, |b|) + 1 \le 101$；空间 $O(L)$。
- 忘记倒序：把最高位当个位，答案完全错误。
- 漏掉最后的进位：$999 + 1$ 会输出 000 或 00。
- 不去前导零：000+000 输出 000。
- 输出循环方向写反：从 0 到末尾会把答案倒着打印。

提交前自测

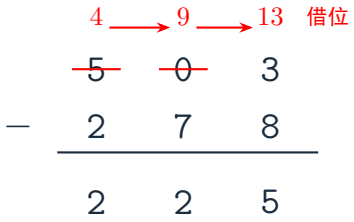
$0 + 0$、$999 + 1$、100 位全 9 加 1、带前导零的输入、两个长度相差很大的数。

加法自测：五组必须过的数据

输入	期望输出	检查点
$0+0$	0	不能输出空行
$9+1$	10	一次进位
$999+1$	1000	连续进位，结果多一位
100 位全 9 加 1	1 后接 100 个 0	长度边界
000123+00456	579	前导零与去零

- 前两组检查最基本的方向与零；第三组检查最高位进位。
- 第四组检查数组长度；第五组检查前导零。
- 再随机生成 18 位以内的数，用 `long long` 暴力对拍。

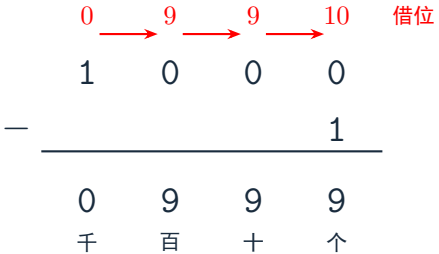
竖式减法：不够减就借 1 当 10



- 适用：两个非负整数，且被减数不小于减数；若 $a < b$，要先交换并处理负号。
- 从个位减起；某一位不够减，就向高位借 1，本位加 10，高位减 1。
- 连续借位时，中间的 0 会变成 9，如上图的十位。

$$d = a_i - borrow - b_i, \qquad \text{若} d < 0, \text{则} d \mathrel{+}= 10,\ borrow = 1.$$

连续借位：$1000-1$ 的借位链



- 个位 $0-1$ 不够减，向十位借；十位也是 0，继续向百位、千位借。
- 千位的 1 借出后变成 0，百位、十位各变成 9，个位变成 10。
- 最后 $10-1=9$，结果是 0999，去掉前导零后输出 999。

易错

中间的 0 借位后是 9，不是 10；只有真正落位的那一位才加 10。

减法适用条件：先比较大小



- 位数不同：位数多的数更大。
- 位数相同：从最高位往低位比，第一个不同的位决定大小。
- 题目保证 $a > b$ 时可直接减；否则先交换，输出负号，再算 $|a - b|$。
- 结果可能为 0，此时输出 0，不要输出空行或 -0。

减法模板：borrow 变量

```cpp
int borrow = 0; // 向高位的借位
vector<int> C;
for (int i = 0; i < (int)A.size(); ++i) {
    int d = A[i] - borrow;
    if (i < (int)B.size()) d -= B[i];
    if (d < 0) {
        d += 10; // 借 1 当 10
        borrow = 1;
    } else {
        borrow = 0;
    }
    C.push_back(d);
}
```

- 调用前要保证 $A \geq B$，否则最高位会继续借，结果没有意义。
- 循环结束后用 `trim` 去掉高位的 0。
- 时间 $O(n)$，空间 $O(n)$。

减法手推：$503-278$ 与 $1000-1$

$503-278$

个位 $3-8$ 不够，向十位借；十位是 0，再向百位借。
百位 $5\to 4$，十位 $0\to 9$，个位 $3\to 13$。
$13-8=5$，$9-7=2$，$4-2=2$，所以 $503-278=225$。

$1000-1$

个位 $0-1$ 不够，借位一路传到千位：$1\to 0$，三个 0 依次变成 $9,9,10$。
$10-1=9$，所以结果是 0999，去前导零后为 999。

- 检查方法：差 $+$ 减数 $=$ 被减数。$225+278=503$，$999+1=1000$。

减法易错点与自测

- 忘记减去 borrow：上一位借走的值没有参与当前位。
- 连续借位时把 0 直接当成 10：中间的 0 借出后应剩 9。
- 结果高位留下 0：$503 - 278$ 可能输出 0225。
- 被减数小于减数：没有交换，最高位借位后结果错误。
- 结果为 0 时输出 -0 或空行。

自测数据

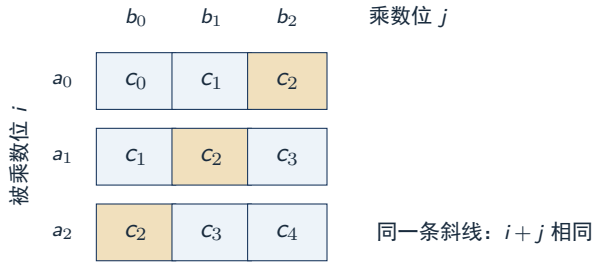
$0-0$、$9-9$、$1000-1$、$100000-99999$、$123456-123455$、带前导零的输入、两个数相等。

竖式乘法：逐位相乘，错位相加

	1	2	3	
×		4	5	
	6	1	5	123×5
4	9	2	0	123×4，左移一位
5	5	3	5	相加得 5535

- 适用：非负整数；普通竖式乘法的时间是 $O(nm)$。
- 乘数第 j 位去乘被乘数第 i 位，结果落在 10^{i+j} 这一位。
- 乘数第 j 位的部分积要左移 j 位，也就是末尾补 j 个 0。

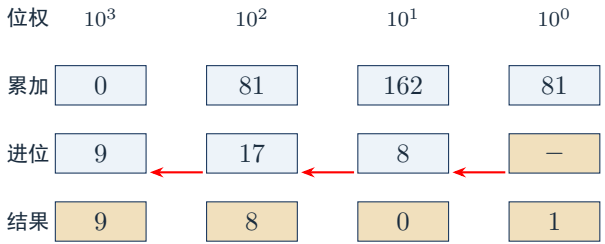
对位矩阵：$a_i b_j$ 落在 c_{i+j}



$$c_{i+j} \ += \ a_i \times b_j$$

- 表格中同一格可能被多次累加，所以 c 要初始化为 0。
- 先全部累加，最后统一从低位向高位处理进位。
- 结果最多 $n+m$ 位，数组开到 $n+m$ 或 $n+m+1$。

先累加、后进位：99×99 的例子



- 原始累加结果是 $c = [81, 162, 81]$，它们不是合法的十进制位。
- 从低位统一进位：$81 \to$ 写 1 进 8；$162 + 8 = 170 \to$ 写 0 进 17。
- $81 + 17 = 98 \to$ 写 8 进 9；最高位补 9，得到 9801。

乘法模板（1/2）：对位累加

```
vector<int> C(A.size() + B.size(), 0);
for (int i = 0; i < (int)A.size(); ++i) {
    for (int j = 0; j < (int)B.size(); ++j) {
        C[i + j] += A[i] * B[j]; // 对位累加
    }
}
```

- 双重循环枚举每一对 (i, j)，把乘积加到 c_{i+j}。
- 此时 C 中的数可能远大于 9，先不要取模。
- 数组长度取 $n + m$ 就够；若担心最高位进位，可以多开一位。

乘法模板（2/2）：统一进位

```
int carry = 0;
for (int i = 0; i < (int)C.size(); ++i) {
    int cur = C[i] + carry;
    C[i] = cur % 10; // 本位
    carry = cur / 10; // 进位
}
while (C.size() > 1 && C.back() == 0) {
    C.pop_back();
}
```

- 进位必须从低位向高位走，和加法一样。
- 最高位进位后，C 的长度可能增加，所以循环用 `C.size()`。
- 最后去掉高位的 0；若两个数都是 0，保留一个 0。

高精 × 低精：乘方、阶乘都要用

```
vector<int> mulSmall(const vector<int>& a, int b) {
    vector<int> c;
    long long carry = 0;
    for (int i = 0; i < (int)a.size() || carry; ++i) {
        long long digit = (i < (int)a.size()) ? a[i] : 0;
        long long cur = carry + digit * b;
        c.push_back(cur % 10);
        carry = cur / 10;
    }
    while (c.size() > 1 && c.back() == 0) {
        c.pop_back();
    }
    return c;
}
```

- 适用：大整数乘小整数；中间结果用 `long long`，防止 $a_i \cdot b$ 溢出。

例 1.3 | 高精度乘法：题意与手推

题意

输入两个小于 100 位的正整数 M, N，输出 $M \times N$。

样例 36×3：

$6 \times 3 = 18 \Rightarrow$ 写8 进1; $3 \times 3 + 1 = 10 \Rightarrow$ 写0 进1; 最高位补1 $\Rightarrow 108.$

再手推 99×99：

$$c_0 = 81, \quad c_1 = 9 \times 9 + 9 \times 9 = 162, \quad c_2 = 81.$$

进位后：$81 \to$ 写 1 进 8；$162 + 8 = 170 \to$ 写 0 进 17；$81 + 17 = 98 \to$ 写 8 进 9；最高位 9，结果 9801。

- 思路：双重循环累加 c_{i+j}，再统一进位。
- 注意中间位 $c_1 = 162$，远大于 9，不能边乘边只进一位。

例 1.3 | 完整程序（1/3）：读入与倒序

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

int main() {
    string sa, sb;
    cin >> sa >> sb; // 读入两个字符串
    vector<int> A, B;
    for (int i = (int)sa.size() - 1; i >= 0; --i) {
        A.push_back(sa[i] - '0');
    }
    for (int i = (int)sb.size() - 1; i >= 0; --i) {
        B.push_back(sb[i] - '0');
    }
    vector<int> C(A.size() + B.size(), 0);
```

下一页开始对位累加；C 的长度取两个因数长度之和。

例 1.3 | 完整程序（2/3）：对位累加

```
for (int i = 0; i < (int)A.size(); ++i) {
    for (int j = 0; j < (int)B.size(); ++j) {
        C[i + j] += A[i] * B[j]; // 对位累加
    }
}
int carry = 0;
for (int i = 0; i < (int)C.size(); ++i) {
    int cur = C[i] + carry;
    C[i] = cur % 10; // 本位
    carry = cur / 10; // 进位
}
```

先累加、后进位；两步之间不要提前取模。

例 1.3 | 完整程序（3/3）：去零与输出

```
    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;
}
```

- 输出前必须去掉高位 0；否则 36×3 可能输出 0108。

例 1.3 | 复杂度与易错点

- 时间 $O(nm)$，空间 $O(n+m)$；$n, m < 100$ 时非常轻松。
- 数组只开 $n+m$：最高位进位无处可放，答案少一位。
- 边乘边只保留一位、丢掉 c_{i+j} 的大数：中间位会算错。
- 忘记把 C 清零：累加会带上随机初值。
- 结果是 0 时（若有 0 输入），去前导零后要保留一个 0。

高精度乘法（II） 的提醒

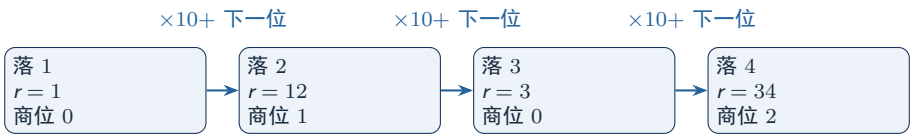
若两个数各有 200000 位，$O(nm)$ 约 4×10^{10} 次运算，必然超时；那时要换 NTT/FFT，见本章最后一节。

乘法自测：0、进位、错位

输入	期望输出	检查点
0×12345	0	结果为 0，保留一位
99×99	9801	中间位 162 的进位
123×45	5535	部分积错位相加
999×9	8991	连续进位
10×10	100	末尾的 0 不能被误删

- 第三组专门检查“左移一位”有没有写对。
- 第五组检查去前导零时只删高位，不删末尾的有效 0。
- 用 18 位以内的随机数对拍 `long long` 乘法。

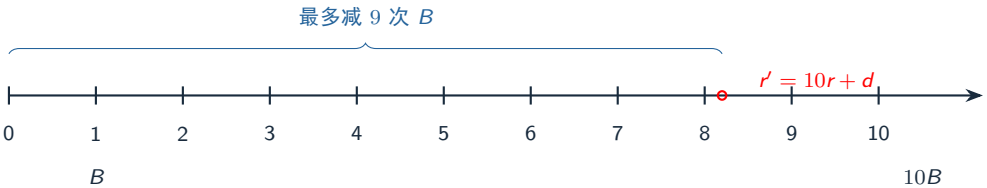
长除法：从最高位开始，逐位取商



商 $0102 \to 102$，余 10

- 适用：非负整数，除数不为 0；按十进制逐位处理。
- 每次“落下”一位：$r' = 10r + d$，再决定这一位的商。
- 这一位的商是 $q = \lfloor r'/B \rfloor$，新余数是 $r' \bmod B$。
- 商的最高位可能是 0，输出前统一去掉，但至少留一位。

试商：为什么一位商最多是 9



- 落位前余数满足 $r < B$，所以 $r' = 10r + d < 10B$。
- 因此这一位的商最多是 9，不会出现两位商。
- 从 0 到 9 试：能减就减一次 B，商位加 1。
- 若 $r' < B$，这一位的商就是 0，余数不变。

效率

每次试商最多减 9 次，每次减法 $O(m)$，所以一位的代价是 $O(10m)$，总时间 $O(10nm)$。

比较大小：先比长度，再逐位比

```cpp
int cmpBig(const vector<int>& a, const vector<int>& b) {
    if (a.size() != b.size()) { // 位数不同
        return a.size() < b.size() ? -1 : 1;
    }
    for (int i = (int)a.size() - 1; i >= 0; --i) {
        if (a[i] != b[i]) {
            return a[i] < b[i] ? -1 : 1; // 高位先比
        }
    }
    return 0; // 完全相等
}
```

- 倒序存储下，最高位在 `size()-1`，所以从末尾往前比。
- 返回 $-1, 0, 1$，分别表示小于、等于、大于，方便调用。

大整数减法：除法的内层操作

```
vector<int> subBig(const vector<int>& a, const vector<int>& b) {
    vector<int> c;
    int borrow = 0;
    for (int i = 0; i < (int)a.size(); ++i) {
        int d = a[i] - borrow;
        if (i < (int)b.size()) d -= b[i];
        borrow = (d < 0) ? 1 : 0; // 是否借位
        if (d < 0) d += 10; // 借 1 当 10
        c.push_back(d);
    }
    while (c.size() > 1 && c.back() == 0) {
        c.pop_back();
    }
    return c;
}
```

- 调用者保证 $a \geq b$；结果要去前导零，否则比较大小会出错。

高精除主循环：落位、试商、减

```
vector<int> q; // 商，高位在前
vector<int> rem(1, 0); // 余数，倒序存放
for (int i = (int)A.size() - 1; i >= 0; --i) {
    rem.insert(rem.begin(), A[i]); // rem = rem*10 + A[i]
    while (rem.size() > 1 && rem.back() == 0) {
        rem.pop_back();
    }
    int digit = 0;
    while (cmpBig(rem, B) >= 0) {
        rem = subBig(rem, B);
        ++digit;
    }
    q.push_back(digit);
}
```

- `insert(begin(), A[i])` 把新数字放到最低位，相当于 $\times 10+$ 当前位。
- 试商最多 9 次；q 的每一位都要保存，包括开头的 0。

例 1.5 | 高精除：题意与结果校验

题意

输入两个低于 300 位的正整数，输出它们的商和余数，各占一行。

- 样例被除数约 76 位，除数约 61 位，商 15 位，余数 60 位。
- 不要求手算这组大数；用等式 商 $\times$ 除数 $+$ 余数 $=$ 被除数 检查。
- 先用手推小例对齐步骤：$1234 \div 12 = 102$ 余 10。
- 思路：十进制逐位长除；余数每次落一位，试商 $0 \sim 9$，再减去除数。

关键顺序

先落位、再去前导零、再试商；商的每一位都要保存，包括开头的 0。

例 1.5 | 程序（1/6）：字符串转数组

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

vector<int> toDigits(const string& s) {
    vector<int> d;
    for (int i = (int)s.size() - 1; i >= 0; --i) {
        d.push_back(s[i] - '0');
    }
    while (d.size() > 1 && d.back() == 0) {
        d.pop_back();
    }
    return d;
}
```

这个函数把字符串变成倒序数组，并顺手去掉前导零。

例 1.5 | 程序（2/6）：比较大小

```
int cmpBig(const vector<int>& a, const vector<int>& b) {
    if (a.size() != b.size()) {
        return a.size() < b.size() ? -1 : 1;
    }
    for (int i = (int)a.size() - 1; i >= 0; --i) {
        if (a[i] != b[i]) {
            return a[i] < b[i] ? -1 : 1;
        }
    }
    return 0;
}
```

返回 -1 表示 $a < b$，0 表示相等，1 表示 $a > b$。

例 1.5 | 程序（3/6）：大整数减法

```
vector<int> subBig(const vector<int>& a, const vector<int>& b) {
    vector<int> c;
    int borrow = 0; // 向高位的借位
    for (int i = 0; i < (int)a.size(); ++i) {
        int d = a[i] - borrow;
        if (i < (int)b.size()) d -= b[i];
        borrow = (d < 0) ? 1 : 0; // 是否借位
        if (d < 0) d += 10; // 借 1 当 10
        c.push_back(d);
    }
    while (c.size() > 1 && c.back() == 0) {
        c.pop_back();
    }
    return c;
}
```

例 1.5 | 程序（4/6）：主函数上半

```
int main() {
    string sa, sb;
    cin >> sa >> sb;
    vector<int> A = toDigits(sa);
    vector<int> B = toDigits(sb);
    if (cmpBig(A, B) < 0) { // 被除数 < 除数
        cout << 0 << '\n';
        for (int i = (int)A.size() - 1; i >= 0; --i) {
            cout << A[i];
        }
        cout << '\n';
        return 0;
    }
    vector<int> q; // 商，高位在前
    vector<int> rem(1, 0); // 余数，倒序存放
```

被除数小于除数时，商为 0，余数就是被除数，单独处理。

例 1.5 | 程序（5/6）：逐位长除

```
for (int i = (int)A.size() - 1; i >= 0; --i) {
    rem.insert(rem.begin(), A[i]); // 落一位
    while (rem.size() > 1 && rem.back() == 0) {
        rem.pop_back();
    }
    int digit = 0;
    while (cmpBig(rem, B) >= 0) {
        rem = subBig(rem, B);
        ++digit;
    }
    q.push_back(digit);
}
int pos = 0;
while (pos + 1 < (int)q.size() && q[pos] == 0) {
    ++pos; // 跳过商的前导零
}
```

例 1.5 | 程序（6/6）：输出商与余数

```
    for (int i = pos; i < (int)q.size(); ++i) {
        cout << q[i];
    }
    cout << '\n';
    for (int i = (int)rem.size() - 1; i >= 0; --i) {
        cout << rem[i];
    }
    cout << '\n';
    return 0;
}
```

- 商从第一个非零位 pos 开始输出；若商为 0，pos 停在最后一位。
- 余数按倒序存储，输出时同样从末尾走到下标 0。

高精除低精：除以 13 的写法

```
vector<int> divSmall(const vector<int>& a, int b, int& rem) {
    vector<int> q(a.size());
    long long r = 0; // 当前余数
    for (int i = (int)a.size() - 1; i >= 0; --i) {
        r = r * 10 + a[i]; // 落下一位
        q[i] = (int)(r / b); // 这一位的商
        r %= b; // 新余数
    }
    while (q.size() > 1 && q.back() == 0) {
        q.pop_back();
    }
    rem = (int)r;
    return q;
}
```

- 适用：除数能用 int 存下；q 仍然按倒序存储。
- 从最高位开始，r 始终小于 b；不需要试商循环，直接除即可。

除法易错点与自测

- 商的前导零：第一位可能是 0，输出前要去掉，但至少留一位。
- 余数恰好为 0：输出 0，不能输出空行。
- 被除数小于除数：商为 0，余数就是被除数，要单独处理。
- 余数落位时忘了先插入新数字：相当于少算一位，商整体错位。
- 试商时比较函数写错：位数相同却没有从高位比，商位会偏大或偏小。

检查公式

商 $\times$ 除数 $+$ 余数 $=$ 被除数，且 $0 \le$ 余数 $<$ 除数。

N 进制：每一位的权重

$$2A6_{16} \quad \boxed{2} \quad \boxed{\texttt{A}} \quad \boxed{6} \quad = 2 \times 256 + 10 \times 16 + 6 = 678$$

16^2 16^1 16^0

- N 进制逢 N 进 1，每一位的数字范围是 $0 \sim N-1$。
- 数值 $= \sum d_i N^i$，下标 0 仍然是个位。
- 十进制用 $0 \sim 9$；十六进制还要用 `A`$\sim$`F` 表示 $10 \sim 15$。
- 本题的进制 N 满足 $2 < N \leq 10$ 或 $N = 16$，字符只有这两类。

重点

所有加法都在 N 进制下进行：逢 N 进 1，不是逢 10 进 1。

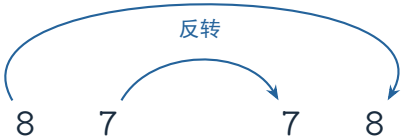
字符与数字：0∼9 和 A∼F

```
int charToDigit(char ch) {
    if (ch >= '0' && ch <= '9') {
        return ch - '0';
    }
    return toupper((unsigned char)ch) - 'A' + 10;
}
```

```
char digitToChar(int d) {
    if (d < 10) {
        return char('0' + d);
    }
    return char('A' + d - 10);
}
```

- 读入时大小写都可能出现，用 toupper 统一成大写。
- 输出时数字 $10 \sim 15$ 要写回 A∼F，不能直接输出整数。

回文判断与反转



$87 + 78 = 165$，不是回文，继续

- 回文：从左读与从右读相同；一位数一定是回文。
- 一步操作：把当前数倒过来，与自身相加（按 N 进制）。
- 重复直到回文；若 30 步还没得到，输出 `Impossible`。

边界

一开始就是回文时，答案是 0 步，不是 1 步。

回文判断的代码

```cpp
bool isPalindrome(const vector<int>& a) {
    int i = 0;
    int j = (int)a.size() - 1;
    while (i < j) {
        if (a[i] != a[j]) {
            return false; // 对称位不同
        }
        ++i;
        --j;
    }
    return true;
}
```

- 双指针从两端向中间走，最多比较 $\lfloor n/2 \rfloor$ 次。
- 倒序存储不影响回文判断：a[0] 与 a[n-1] 比较即可。
- 空数组和一位数都返回 true。

例 1.6 | 回文数（Noip1999）：题意与手推

题意

给定进制 N 和 N 进制数 M，求最少几步能得到回文数；30 步内不行则输出 `Impossible`。

样例：$N = 9,\ M = 87_9$，六步得到回文数：

$$87 \to 176 \to 857 \to 1726 \to 8107 \to 16126 \to 78287.$$

- 每一步都是“原数 $+$ 反转数”，并按 9 进制进位。
- 78287 从左读和从右读相同，所以答案是 6。
- 思路：数组倒序存 N 进制各位 $\to$ 判断回文 $\to$ 反转相加 $\to$ 计步。

例 1.6 | 程序（1/3）：N 进制加法

```
int base;
vector<int> addBase(const vector<int>& a, const vector<int>& b) {
    vector<int> c;
    int carry = 0;
    int n = max(a.size(), b.size());
    for (int i = 0; i < n || carry; ++i) {
        int s = carry;
        if (i < (int)a.size()) s += a[i];
        if (i < (int)b.size()) s += b[i];
        c.push_back(s % base); // 本位
        carry = s / base; // 逢 base 进 1
    }
    return c;
}
```

- 进位条件是 `s / base`，不是固定的 10。
- 返回的数组已经按倒序存放，不需要再反转。

例 1.6 | 程序（2/3）：主循环

```
int main() {
    string mstr;
    cin >> base >> mstr;
    vector<int> m;
    for (int i = (int)mstr.size() - 1; i >= 0; --i) {
        char ch = mstr[i];
        if (ch >= '0' && ch <= '9') {
            m.push_back(ch - '0');
        } else {
            m.push_back(toupper((unsigned char)ch) - 'A' + 10);
        }
    }
    while (m.size() > 1 && m.back() == 0) {
        m.pop_back();
    }
```

读入时把字符转成数字；下一页进入 30 步循环。

例 1.6 | 程序（3/3）：判断与输出

```
    for (int step = 0; step <= 30; ++step) {
        if (isPalindrome(m)) {
            cout << step << '\n';
            return 0;
        }
        if (step == 30) {
            break;
        }
        vector<int> r(m.rbegin(), m.rend());
        m = addBase(m, r);
    }
    cout << "Impossible\n";
    return 0;
}
```

- 循环写成 `step <= 30`：先判断第 30 步是否已经是回文，再决定放弃。
- 如果一开始就是回文，第一次判断就输出 0。

例 1.6 | 30 步、Impossible 与易错点

- 循环写成 `step <= 30`：先判断第 30 步是否已经是回文，再决定放弃。
- 一开始就回文：输出 0，不能先加一次再判断。
- 十六进制字母大小写都要能读入；数字 $10 \sim 15$ 不要误当字符。
- 进位基数写死成 10：九进制、十六进制都会算错。
- 反转后前导零：例如 10_2 反转是 01_2，数值仍是 1，判断回文要看有效位。

经典数据

十进制 196 在 30 步内得不到回文数；提交前用 `10 196` 测一次，应输出 `Impossible`。

五个最常见的错误

1. 方向错：没有倒序存储，或输出时从低位开始打印。
2. 进位丢：加法循环漏掉最后的 `carry`，乘法数组只开 $n+m$。
3. 前导零：不删高位的 0，或删得把唯一的 0 也删掉。
4. 借位错：减法忘记连续借位，把中间的 0 当成 10 直接算。
5. 类型错：该用高精度却用 `long long`，或者用 `double` 近似。

统一检查

每个模板都问自己：数组下标 0 是不是个位？循环有没有覆盖最高位进位？输出是不是从后往前？

调试方法：小数据手算与对拍

- 先用手算验证：$0+0$、$9+1$、$99+1$、$1000-1$、99×99。
- 用 18 位以内的数据写一个 `long long` 暴力程序，与高精度程序随机对拍。
- 边界表：最小值、最大值、全 9、全 0、前导零、长度相差 1、刚好整除。
- 出错时先打印数组（倒序），确认每一位的数值和长度，再查进位。

推荐顺序

先保证加法与减法全对，再写乘法；除法最后写，因为它依赖比较和减法。

模板速查与复杂度

运算	核心式子	时间	结果长度
加法	$c_i = (a_i + b_i + \text{carry}) \bmod 10$	$O(n)$	$\max(n, m) + 1$
减法	不够减则借 1 当 10	$O(n)$	n
乘法	$c_{i+j} \mathrel{+}= a_i b_j$	$O(nm)$	$n + m$
除法	$r = 10r + d$，试商 $0 \sim 9$	$O(10nm)$	n

- 所有模板都先倒序存储，最后去掉前导零再倒序输出。
- 结果长度按上表开数组，宁可多开一位，不要刚好卡边界。

背下来

加法看进位，减法看借位，乘法看对位，除法看余数。

扩展：压位

- 压位：一个数组元素存 9 位十进制（以 10^9 为基数），长度降为约 $1/9$。
- 加法运算量也降为约 $1/9$；乘法是 $O(nm)$，运算量降为约 $1/81$。
- 阶乘 $10000!$ 有 35660 位，用压位后只需约 4000 个元素。
- 压位后进位阈值变成 10^9，乘法中间结果要用 `long long`。
- 输出时除最高位外，其余每一位都要补足 9 位前导零。

适用场景

位数达到 $10^5 \sim 10^6$、又必须用 $O(nm)$ 或 $O(n)$ 算法时，压位能显著减少常数。

扩展：超大整数乘法与 NTT

- 两个 200000 位数相乘，$O(nm)$ 约 4×10^{10}，必须换算法。
- **NTT**：把乘法变成多项式卷积，时间 $O(n \log n)$；模数 998244353 支持足够长的变换。
- 本题每位乘积之和最大约 $9 \times 9 \times 200000 = 1.62 \times 10^7$，小于模数，一次 NTT 就够。
- 若系数和可能超过模数，要用“拆系数”或多次 NTT 再合并。

对应练习

最后一题“高精度乘法（II）”就是 200000 位乘法，普通双重循环会超时，需要用 NTT/FFT。

练习安排：从加减到综合

1. 先做“大整数加法、大整数减法”，把倒序、进位、借位练熟。
2. 再做“计算 2 的 N 次方、求 10000 以内 n 的阶乘、阶乘和”，练习大整数乘小数。
3. 接着做“大整数乘法、除以 13”，掌握乘法对位与逐位取商。
4. 最后做“大整数的因子、高精度乘法（II）”，前者练取模，后者挑战 NTT。

每题的笔记

写四行：存储方式、核心循环、进位/借位规则、边界数据；写完先用样例，再测全 9 和 0。

本章练习题索引 | 1–5 / 9

1. P55040 | 大整数加法（一本通 1168）
2. 大整数减法（一本通 1169） P4973
3. 计算 2 的 N 次方（一本通 1170） P4974
4. 大整数的因子（一本通 1171） P4975
5. 求 10000 以内 n 的阶乘（一本通 1172） P4976

蓝色题名可点击进入学校 OJ；标“题包”的题目在课程题包中提交。

本章练习题索引 | 6–9 / 9

6. P55036 | 阶乘和（一本通 1173）
7. 大整数乘法（一本通 1174） P4977
8. 除以 13（一本通 1175） P4978
9. 高精度乘法（II）（一本通 2128） P4979

至此本章 9 道练习题全部列出；先易后难，做完再回看模板。