# [8] 模拟与高精度

深入浅出程序设计竞赛
第 2 部分 – 初涉算法
V 2021-02

洛谷
www.luogu.com.cn

# 版权声明



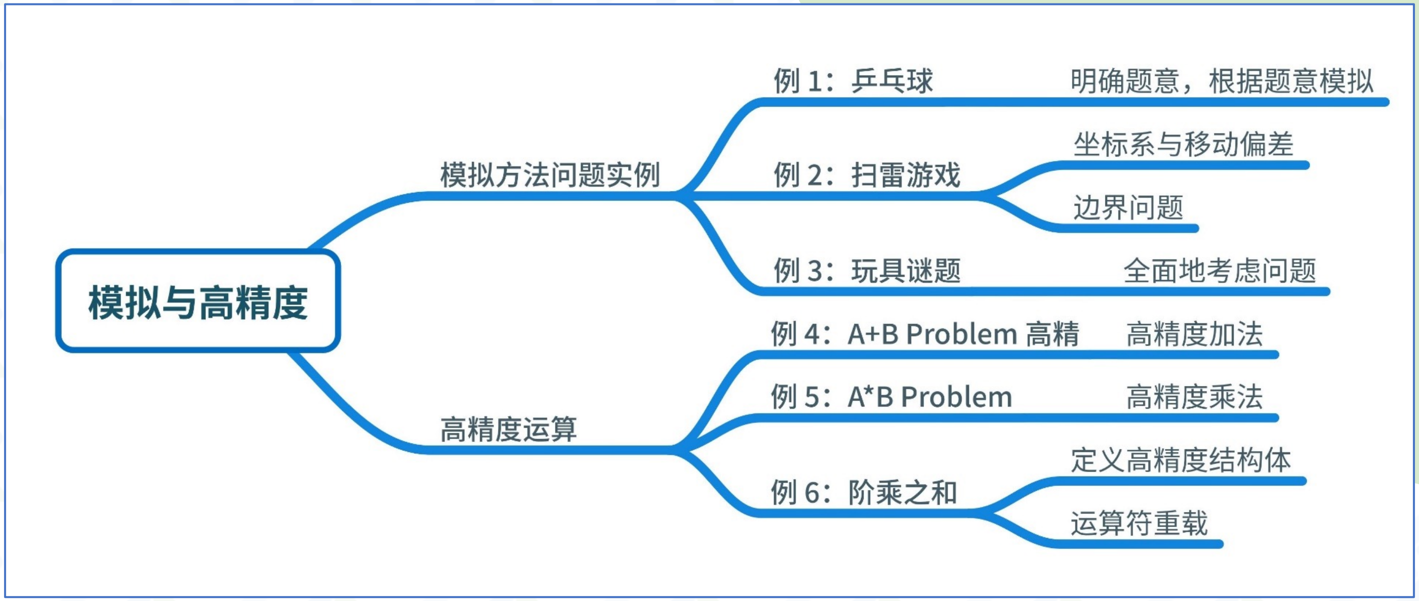
- 其它《深基》配套资源、购买本书等请参阅：https://www.luogu.com.cn/blog/kkksc03/IPC-resources
- 如果课件有任何错误，请在这里反馈 https://www.luogu.com.cn/discuss/show/296741

# 说明

在之前的章节中，我们已经熟悉了C++编程语言的基本使用。从本章开始，我们将接触一些简单的算法，尝试完成一些更有挑战性的问题。

限于篇幅，我们将不再完整呈现程序清单。若有需要，请对照《深入浅出程序设计竞赛 基础篇》查看题目完整代码。

# 本章知识导图


模拟与高精度
模拟方法问题实例
例 1：乒乓球
明确题意，根据题意模拟
例 2：扫雷游戏
坐标系与移动偏差
边界问题
例 3：玩具谜题
全面地考虑问题
高精度运算
例 4：A+B Problem 高精
高精度加法
例 5：A*B Problem
高精度乘法
例 6：阶乘之和
定义高精度结构体
运算符重载

# 第 8 章 模拟与高精度

# 模拟方法问题实例

计算机最大的优点，即擅长机械、重复、批量地按照特定的规则执行任务。
而我们则是利用计算机的这一特点，来模拟现实世界当中的问题

请翻至课本 P117

# 乒乓球

例8.1 （洛谷 P1042，NOIP 2003 普及组）

华华和朋友打球，得到了每一球的输赢记录：W表示华华得一分，L表示对手得一分，E表示比赛信息结束。

一局比赛刚开始时比分为 0:0。每一局中达到 11 分或者 21 分（赛制不同），且领先对方 2 分或以上的选手可以赢得这一局。

现在要求输出在 11 分制和 21 分制两种情况下的每一局得分。

输入数据每行至多有 25 个字母，最多有 2500 行。

```
WWWWWWWWWWWWWWWWWWWW
WWLWE
```

```
11:0
11:0
1:1

21:0
2:1
```

# 乒乓球

第一步，我们需要接受输入。

此处使用一个数组 a 记录每一局的胜负。

```
while(1) {
    cin >> tmp; // 不断读入结果
    if(tmp == 'E') break;
    else if(tmp == 'W') a[n++] = 1; // 华华赢
    else if(tmp == 'L') a[n++] = 0; // 华华输
    // 依照题意，若不属于这三种字符，则应忽略
}
```

# 乒乓球

第二步，我们需要对于每一种赛制、每一局均进行统计，因此需要引入循环：

```
// 两种赛制循环
for (int k = 0; k < 2; k++) {
    int w = 0, l = 0;
    for (int i = 0; i < n; i++ ) {
        w += a[i]; l += 1 - a[i];
        // 这里还需要判断一局比赛是否结束
        // 具体怎么做呢？
    }
    // 未完成的比赛也要输出结果
    cout << w << ":" << l << endl;
    cout << endl;
}
```

# 乒乓球

最关键的第三步，我们考虑将语言描述的规则转化为代码。

题目已经清晰地描述了规则：

1）达到分数下限

2）至少超越对手2分

则一局结束；输出比分。

转化为C++代码（片段）：

```cpp
// 获胜者超过对应分数且超出对手2分
if((max(w, l) >= f[k]) && abs(w - l) >= 2)
{
    cout << w << ":" << l << endl;
    w = l = 0;  // 比分清空，进入下一句
}
```

# 乒乓球

需要注意的细节：

1. 数组要开够，至少需要容纳 25×2500 条得分记录。
2. 读到 E 就停止读入了，后面即使还有内容也都忽略掉。同时遇到换行符或其他字符等也要忽略。
3. 最后还要输出正在进行中的比赛，就算是刚刚完成一局也要输出 0:0。

替换练习：

是否可以不借助数组完成这个任务？

# 小提示：阅读理解

题目（除了背景故事外的）几乎每一句话都很关键，所以一定要**认真审题**！

当你在题目中看到数字、斜体字母等**数学符号**甚至**公式**时，往往意味着它是关键题意信息。在一些场合下，当你看到题目中出现**等宽字体**字母时，往往意味着你的代码里也需要声明这个变量。

有些题目可能存在一些不严谨的地方造成歧义，所以在比赛中如果发现字面意思不清楚，可以找比赛组织者（监考老师、志愿者、答疑帖等）**明确题意**。

# 扫雷游戏

例8.2 （洛谷 P2670，NOIP 2015 普及组）

给出一个 $n \times m$ 的网格，有些格子埋有地雷。求问这个棋盘上每个没有地雷的格子周围（上下左右和斜对角）的地雷数。

输入：第一行两个整数 $n$ 和 $m$ 表示网格的行数和列数，接下来 $n$ 行 $m$ 列的字符矩阵，描述地雷分布情况。字符*表示相应格子是地雷格，字符?表示相应格子是非地雷格。$m$ 和 $n$ 不超过 100。

输出：包含 $n$ 行，每行 $m$ 个字符，描述整个雷区。用*表示地雷格，用周围的地雷个数表示非地雷格。

```
3 3
*??
???
?*?
```

```
*10
221
1*1
```

# 扫雷游戏

同样地，我们需要先输入数据；由于地图是2维的，因此我们需要借助二维数组。我们也注意到需要对每一个格子进行计算，一个二重循环的代码框架就应运而生。

此处我们直接研究如何将逻辑转化为代码。

依题意，对于格子(x, y)，我们需要讨论的坐标为：

| | | |
|---|---|---|
| (x-1,y-1) | (x-1,y) | (x-1,y+1) |
| (x,y-1) | (x,y) | (x,y+1) |
| (x+1,y-1) | (x+1,y) | (x+1,y+1) |

# 扫雷游戏

显然，直接手工在代码里编写8个方向的坐标是正确的。但是会引入大量重复的工作。计算机显然比我们更擅长机械工作，我们没有理由不甩锅给它。

| | | |
|---|---|---|
| (x-1,y-1) | (x-1,y) | (x-1,y+1) |
| (x,y-1) | (x,y) | (x,y+1) |
| (x+1,y-1) | (x+1,y) | (x+1,y+1) |

如果我们将相对(x, y)的偏移量（+1, 0, -1）的所有组合不遗漏、不重复地使用数组保存，则只需要在这个数组上循环，即可遍历所有的方向！

```
const int dx[] =  {1, 1, 1, 0, 0, -1, -1, -1};
const int dy[] =  {-1, 0, 1, -1, 1, -1, 0, 1};
```

# 扫雷游戏

在此基础上，逻辑翻译的工作就非常简单了：

```
// 如果当前格子没有地雷
if (g[i][j] != '*') {
    // 地雷计数为0
    int cnt = 0;
    // 枚举8个方向（自身是不算在内的）
    for (int k = 0; k < 8; k++)
        // 如果第k个方向是地雷，则计数+1
        if (g[i + dx[k]][j + dy[k]] == '*') cnt++;
    // 输出地雷计数
    cout << cnt;
}
```

这种化手工为循环的方式是非常常见的。

# 模拟法：从人到计算机

计算机比人聪明吗？

并不，计算机无法理解任何事物含义；在它“看”来一切都只是编码，甚至只是电场。是我们自己赋予了这些电信号意义。

人比计算机聪明吗？

并不，计算机可以在一纳秒之内计算32位二进制整数加减法。而一般的人脑大多只能算2位十进制（若你能算更多，欢迎上《最强大脑》挑战）。

人如何为这些电信号赋予“意义”？

我们必须将我们的语言中抽象的、灵活的描述转化为计算机中具体的、确定性的程序，才能让计算机执行。

# 模拟法：从人到计算机

一个简单的例子：求一个整数序列中最大的整数。

抽象的描述：取所有元素中最大的数，返回。

```
return max(seq);
```

具体的描述：

取序列第一个元素为候选；
对于从第2到n的每一个元素，
若候选较小则候选替换为当前元素
循环结束后，返回候选。

```
int candidate = seq[1];

for (int i = 2; i <= n; i++)
    if (candidate < seq[i])
        candidate = seq[i];

return candidate;
```

计算机无法直接理解并运行抽象的描述，但是可以“逐字逐句”翻译并极其高效地执行具体的描述。请牢记，编程的难点并不是语法（术），而是构造确定性描述的思想（道）。

# 高精度运算

我们已经知道，每一种数据类型的表示能力都是有限的。但表示更大范围的数据的需求将永远存在。

请翻至课本 P121

# A+B Problem （高精）

例 8.4 （洛谷 P1601）

分别在两行内输入两个 500 位以内的十进制非负整数，求他们的和。

样例输入：

```
514
495
```

样例输出：

```
1009
```

# A+B Problem （高精）

500位即$10^{500}$，超过了任何基本数据类型的表示范围。

一个很容易想到的解决方案是使用数组：每一个元素存储一个十进制位。建议使用小端存储（个位放最前面）。

小端

```
int a = 123;
int A[] = {3, 2, 1};
```

大端

```
int a = 123;
int A[] = {1, 2, 3};
```

大端存储虽然看似更直观，但是当处理进位时就会遇到问题：必须要将所有元素移位，为新位腾出空间！

实际上，如果将数组看做多项式$T(10) = \sum A_i 10^i$的系数，则小端存储才是更“直观”的选择。

# A+B Problem （高精）

现在表示了数，如何进行计算呢？

回到数学课的课堂上，我们学过竖式计算：

```
  514
 +495
 1100
-----
 1009
```

此时我们套用模拟问题的思路，可构造一个竖式加法具体的描述：

1. 从低到高位计算；
2. 对于较长数每一位i，和的第i位为两加数第i位与当前进位三者之和；
3. 若和大于10，则下一进位为1，当前位和减10；
4. 结束后，若进位非0，则结果增加一位，值为进位（此处必为1）。

# A+B Problem （高精）

现在将这一描述转化为C++代码*，完成本题：

```
// 从低到高位计算
for (int i = 0; i < len; i++) {
    // 和的第i位为两加数第i位与当前进位三者之和
    c[i] += a[i] + b[i];
    // 若和大于10，则下一进位为1
    c[i + 1] = c[i] / 10;
    // 当前位和减10
    c[i] %= 10;
}
// 结束后，若进位非0，则结果增加一位，值为进位（此处必为1）
if (c[len + 1])
    len++;
```

思考：使用数组0~(n-1)和使用数组1~n在进行高精度运算时，分别有什么么好处？

# 高精度运算

高精度运算可分为以下类别：

- 高精±高/单精：最为简单，在上一题中介绍
- 高精×单精：相对简单，本质上也只需要处理进位
- 高精×高精：较为复杂，此处只要求掌握$O(n^2)$算法
- 高精÷高精：极其复杂，因为根据竖式除法规则，每一次上商都需要进行一次高精×单精与高精±高精；此处暂不要求掌握

# A*B Problem

例 8.5 （洛谷 P1303）

分别在两行内输入两个 2000 位以内的十进制非负整数，求他们的积。

样例输入：

```
514
495
```

样例输出：

```
254430
```

# A*B Problem

我们采用与上例相同的方法，使用数组表示数。

如何进行竖式乘法？

```
    514
  × 495
  -----
   2570
  4626
 2056
 ------
 254430
```

似乎还有一些复杂……

# A*B Problem

更清晰地表示运算过程。注意我们不再立即处理进位，而是让逐位乘法的结果保留在当前列上：

| 数 | 第5位 | 第4位 | 第3位 | 第2位 | 第1位 | 第0位 |
|---|---|---|---|---|---|---|
| a | | | | 5 | 1 | 4 |
| b | | | | 4 | 9 | 5 |
| a*b[0] | | | | 25 | 5 | 20 |
| a*b[1] | | | 45 | 9 | 36 | |
| a*b[2] | | 20 | 4 | 16 | | |
| 中间产物 | | 20 | 49 | 50 | 41 | 20 |
| 处理进位 | 2，进0 | 25，进2 | 54，进5 | 54，进5 | 43，进4 | 20，进2 |
| 结果 | 2 | 5 | 4 | 4 | 3 | 0 |

观察每列不难发现，若不考虑进位，a[i]*b[j] 的贡献全都在中间产物的第i+j位上！因此可以先算出贡献，最后处理进位。

# A*B Problem

至此，我们可以给出最终实现*了：

```
// 计算贡献
for (int i = 0; i < lena; i++)
    for (int j = 0; j < lenb; j++)
        c[i + j] += a[i] * b[j];
// 乘积的位数不超过两数的位数之和
int lenc = lena + lenb;
for (int i = 0; i < lenc; i++) {
    c[i + 1] += c[i] / 10;  // 处理进位
    c[i] %= 10;
}
// 去掉前导零
for (; !c[lenc];)
    lenc--;
```

是的，这里就可以看出使用0~(n-1)项的好处：此时等价于计算两个多项式$T(x) = \sum A_i x^i$的乘法，取当$x = 10$时的特例！避免了因为思考是否需要+1而导致出错的可能性！

# 封装高精度结构

我们已经可以利用高精度运算解决实际问题。但是此时的代码看起来不够简洁。例如，当你阅读上一题的代码时，可能无法第一反应意识到，这是在计算两数的乘积。

是否能通过编写一些自己的“工具箱”，使得高精度运算可以像基础数据类型运算的代码一样简便直观？

答案当然是可以的！

# 封装高精度结构

我们可以使用结构体。首先，我们需要在其中定义与之前类似的数组，作为高精度表示的“身”。

```
struct Bigint {
    int len, a[maxn];
    Bigint(int x = 0) { // 初始化数值为x
        memset(a, 0, sizeof(a));
        for (len = 0; x; len++)
            a[len] = x % 10, x /= 10;
    }
    // 重载[]，可以直接用x[i]代表x.a[i]，编写时更加自然
    int &operator[](int i) { return a[i]; }
    void flatten(int L) { // 处理进位
        len = L;
        for (int i = 0; i < len; i++)
            a[i + 1] += a[i] / 10, a[i] %= 10;
        for (; !a[len];)
            len--;
    }
    void print() {
        for (int i = max(len-1, 0); i >= 0; i--)
            printf("%d", a[i]);  // 若数值为0，也需要输出0
    }
};
```

# 封装高精度结构

之后，再重载我们所需要的运算（例如+、*），实现高精度运算的“心”。

```
// 表示两个 Bigint 类相加，返回一个 Bigint 类
Bigint operator+(Bigint &a, Bigint &b) {
    Bigint c;
    int len = max(a.len, b.len);
    for (int i = 0; i < len; i++)
        c[i] += a[i] + b[i]; // 计算贡献
    // 答案不超过 len+1 位，所以用 len+1 做一遍 “展平” 处理进位。
    c.flatten(len + 1);
    return c;
}
//表示 Bigint 类乘整型变量，返回一个 Bigint 类
Bigint operator*(Bigint &a, int b) {
    Bigint c;
    int len = a.len;
    for (int i = 1; i <= len; i++)
        c[i] = a[i] * b;// 计算贡献
    // int类型最长10位，所以可以这样做一遍 “展平” 处理进位。
    c.flatten(len + 11);
    return c;
}
```

# 封装高精度结构

此时，“身心”合一，万题可破！

例6 阶乘之和（洛谷 1009，NOIP普及组 1998）

用高精度计算出 $S = 1! + 2! + 3! + \cdots + n!\ (n \le 50)$

直接模拟！Bigint的使用就像int一样丝滑！

```
int main() {
    Bigint ans(0), fac(1);
    // 分别用 0 和 1 初始化 ans 与 fac，
    // 如果要将常数赋值给大整数，可以使用ans=Bigint(233) 的办法
    int m;
    cin >> m;
    for (int i = 1; i <= m; i++) {
        fac = fac * i; // 模拟题意
        ans = ans + fac;
    }
    ans.print(); // 输出答案
    return 0;
}
```

# 课后习题与实验

学而时习之，不亦说乎。学而不思则罔，思而不学则殆。——孔子

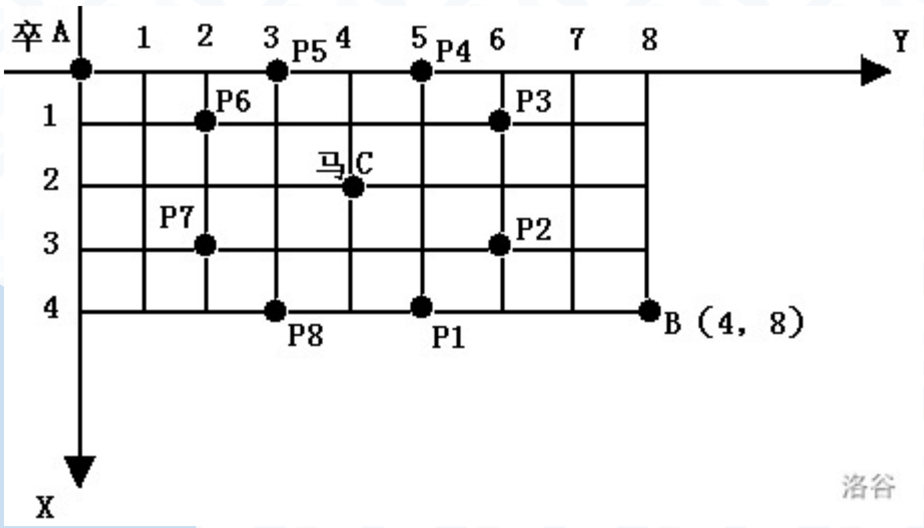
请翻至课本 P125

# 扫雷游戏

本题中相对位移的数组是这样的：

```
const int dx[] =  {1, 1, 1, 0, 0, -1, -1, -1};
const int dy[] =  {-1, 0, 1, -1, 1, -1, 0, 1};
```

替换练习：如果地雷的检定范围是类似国际象棋中的马（骑士），则dx、dy数组的内容应该为什么？

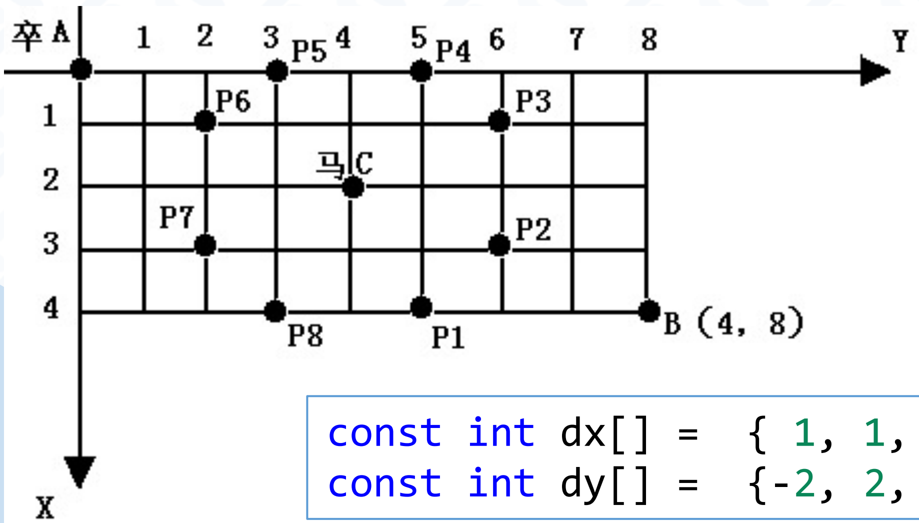


# 扫雷游戏

本题中相对位移的数组是这样的：

```
const int dx[] =  {1, 1, 1, 0, 0, -1, -1, -1};
const int dy[] =  {-1, 0, 1, -1, 1, -1, 0, 1};
```

替换练习：如果地雷的检定范围是类似国际象棋中的马（骑士），则dx、dy数组的内容应该为什么？不重不漏，循序一般不重要。




```
const int dx[] =  { 1, 1, 2,  2, -1, -1, -2, -2};
const int dy[] =  {-2, 2, 1, -1,  2, -2,  1, -1};
```

# 复习

模拟方法 深刻理解题意 转化为具体描述 翻译为程序

抽象的描述 无法执行，但可以定义目的，即“需要做什么”

具体的描述 可以直接翻译为程序，定义方案，即“怎么做”

高精度运算

基本方法：数组模拟 竖式运算 位压缩

高精度加法、高精度乘法

运算符

# 作业

## 习题 8.2 生活大爆炸版石头剪刀布（洛谷 P1328）

有某种剪刀石头布的变种，在传统的石头剪刀布的基础上增加了蜥蜴人和斯波克两种角色，胜负关系如下显示（甲对乙的结果）：

| 甲　乙 | 剪刀 | 石头 | 布 | 蜥蜴人 | 斯波克 |
|---|---|---|---|---|---|
| 剪刀 | 平 | 输 | 赢 | 赢 | 输 |
| 石头 | | 平 | 输 | 赢 | 输 |
| 布 | | | 平 | 输 | 赢 |
| 蜥蜴人 | | | | 平 | 赢 |
| 斯波克 | | | | | 平 |

小A 和 小B 出拳都是有周期性规律的（周期分别是 N_1 和 N_2），玩 N(N≤200) 局，每局胜者得 1 分，败者或平局均不得分。

给出 N,N_1,N_2 和他们出招的规律序列，剪刀、石头、布、蜥蜴人、斯波克分别用 0、1、2、3、4 表示，输出最后两人的得分

# 作业

## 习题 8.3 两只塔姆沃斯牛（洛谷 P1518）

给出个 10×10 的地图，一个格子可能是 .（空地）、 *（障碍）、C（牛的初始位置）、 F（农民初始位置）。开始面正北（上方）。

每分钟他们运行方式：可以向前移动或是转弯。如果前方无障碍（地图边沿也是障碍），它们会按照原来的方向前进一步。否则它们会用这一分钟顺时针转90度。

几分钟后他们会落在同一格里？如果他们永远不相遇输出 0。

```
*...*.....
......*...
...*...*..
..........
...*.F....
*.....*...
...*......
..C......*
...*.*....
.*.*......
```

```
49
```

# 作业

## 习题 8.4 多项式输出（洛谷 P1067）

一元$n$次多项式可表示为 $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, a_n \neq 0$ ；给出一个一元多项式各项的次数和系数，请按照规则（详见教材P127）写出这个多项式。

```
100 -1 1 -3 0 10
```

```
100x^5-x^4+x^3-3x^2+10
```

# 作业

习题 8.8 阶乘数码（洛谷 P1591）

求 $n!\,(n \le 1000)$ 中某个数码 $a(0 \le a \le 9)$ 出现的次数。

例如 $10! = 3628800$ 中，8 出现的次数是 2 次。

习题 8.9 最大乘积（洛谷 P1249）

一个正整数一般可以分为几个互不相同的自然数的和，现在你的任务是将指定的正整数 $n(3 \le n \le 10000)$ 分解成若干个互不相同的自然数的和，且使这些自然数的乘积最大。

输出分解方案和最大乘积。

提示：如果现有的知识不能帮你解决这个问题，请思考数学方法

# 参考阅读材料

以下的内容限于课件篇幅未能详细阐述。如果学有余力，可自行翻阅课本作为扩展学习。

- P119 例 8.3：模拟中讨论多种情况，并将重复情况合并
- 习题 8.5、8.6、8.7、8.10。