

一、三大结构（顺序/选择/循环）

（一）考点分析

结构类型	重点考点	考察频率	难度
顺序结构	变量覆盖陷阱	30%	★
	整数除法截断	25%	★★
	复合表达式求值顺序	20%	★★
	输入输出格式控制	15%	★
	数据范围溢出	10%	★★★★
选择结构	else 匹配原则	35%	★★
	复合条件优化	25%	★★★★
	边界值处理	20%	★★
	switch-case 穿透问题	15%	★★
	浮点数相等判断	5%	★★★★
循环结构	边界值处理	30%	★★
	break/continue 控制流	25%	★★★★
	死循环识别与预防	20%	★★★★
	嵌套循环复杂度分析	15%	★★★★
	循环变量更新时机	10%	★★

（二）核心知识点

1、顺序结构

(1) 变量覆盖陷阱

- 核心知识点：在多步计算中，**变量的值可能被后续操作覆盖**，导致原始数据丢失。
- 黄金法则：**关键变量不可重复使用**，需引入临时变量存储中间结果。
- 高频公式：

```
temp = a; // 保护原始值

a = b + c; // 新值覆盖

result = temp * a; // 使用原始值计算
```

- 易错点：在连续计算中直接覆盖关键变量（如 2014 年真题 $a = d + c$ 覆盖了原始 a ）。
- 程序模板：

```
int a, b, c, d, ans;

cin >> a >> b >> c;

int orig_a = a; // 保护原始 a 值

d = a - b;

a = d + c;      // 覆盖 a

ans = orig_a * b; // 使用原始 a
```

(2) 整数除法截断

- 核心知识点：**int 类型除法会丢弃小数部分**（如 $5/2=2$ ）。
- 黄金法则：任何除法操作**至少有一个操作数转为浮点**。
- 高频公式：

```
double result = 1.0 * a / b;
```

- 易错点：

`sum += 1/n` (应为 `sum += 1.0/n`) 。

- 程序模板：

```
int n;

double sum = 0;

cin >> n;

for(int i=1; i<=n; i++)

    sum += 1.0 / i; // 必须 1.0
```

(3) 复合表达式求值顺序

- 核心知识点：`*` / `%` 优先级高于 `+` - , **= 从右向左** 计算。
- 黄金法则：复杂表达式加括号明确顺序。
- 高频公式：

```
int result = (a + b) * c;
```

- 易错点：

`x = a + b * c` 误算为 `(a+b)*c`。

(4) 输入输出格式控制

- 核心知识点：`cout` 需严格匹配题目要求的空格/换行。
- 黄金法则：输出前复制题目要求的格式字符串。
- 高频公式：

```
cout << a << "+" << b << "=" << a+b;
```

- 易错点：漏写 `<<` 连接符或空格 (2013 年真题) 。

(5) 数据范围溢出

- 核心知识点：`int` 最大 ± 21 亿 ($2.1 * 10^9$) 。

- 黄金法则：超过 10^9 用 long long。
- 高频公式：

```
long long product = (long long)a * b;
```

- 程序模板：

```
long long factorial(int n) {
    long long res = 1;
    for(int i=2; i<=n; i++)
        res *= i; // 10!已超 300 万
    return res;
}
```

2、选择结构

(1) else 匹配原则

- 核心知识点：else 总是匹配最近未匹配的 if。
- 黄金法则：嵌套超过两层必须加{ }。
- 易错点：

```
if(a>b)
    if(b>c) cout<<"A";
    else cout<<"B"; // else 匹配内层 if!
```

- 程序模板：

```
if(cond1) {
    if(cond2) { ... }
} else { // 明确匹配外层 if
    ...
}
```

```
}
```

(2) 复合条件优化

- 核心知识点: **&&**短路求值 (**前假后不执行**) , **||**前真后不执行。
- 黄金法则: 高概率条件放前面, 避免冗余计算。
- 高频公式:

```
if(b != 0 && a/b > 2) // 避免除 0 错误
```

```
if(is_valid || time_consuming_check()) // 高效条件排序
```

(3) 边界值处理

- 核心知识点: 区间端点需单独验证, 比如 0-9 之间是数字, a-z 之间是小写字母, A-Z 之间是大写字母, 100-999 之间是三位数。
- 黄金法则: 测试 $x=0, 1, INT_MAX$ 边界值。
- 高频公式: `if(score >= 90)` // 包含 90 分
- 程序模板:



```
if(x >= low && x <= high) {
```

```
// 区间包含端点
```

```
}
```

 $a < x \leq b$

 $[a, b]$

(4) switch-case 穿透问题

- 核心知识点: **缺少 break** 会继续执行后续 case。
- 黄金法则: 非刻意穿透**必须写 break**。
- 高频公式:

```
switch(grade) {
```

```
case 'A': score=90; break; // 必须 break
```

```
case 'B': score=80; break;
```

```
default: score=0;
```

```
}
```

(5) 浮点数相等判断

- 核心知识点：浮点精度误差导致 == 失效。
- 黄金法则：用误差范围替代直接相等判断。
- 高频公式：

```
const double EPS = 1e-6;
```

```
if(fabs(a - b) < EPS) // 视为相等
```

3、循环结构

(1) 边界值处理

- 核心知识点：for(i=0; i<=n; i++) 比 i<n 多执行 1 次。
- 黄金法则：验证 i=0 和 i=n-1 的情况。
- 高频公式：

```
for(int i=0; i<n; i++) // 执行 n 次
```

```
for(int i=1; i<=n; i++) // 执行 n 次
```

(2) break/continue 控制流

- 核心知识点：break 退出整个循环，continue 跳过本次剩余代码。
- 黄金法则：while 循环中用 continue 前需更新循环变量。
- 易错点：

```
while(i<n) {
```

```
    if(cond) continue; // 可能跳过 i++ 导致死循环
```

```
    i++;
```

```
}
```

(3) 死循环识别与预防

- 核心知识点：循环条件**必须能变为假**。
- 黄金法则：设置**最大循环次数**保险。
- 高频公式：

```
for(int i=0; i<MAX_ITER; i++) {
    if(exit_cond) break;
}
```

(4) 嵌套循环复杂度分析

- 核心知识点：三重循环 $O(n^3)$ 最好只处理 $n < 200$ 的数据。
- 黄金法则： 10^6 次操作 ≈ 1 秒（CSP-J 环境）。
- 高频公式：

```
// O(n^2)算法 n≤5000
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
```

(5) 循环变量更新时机

- 核心知识点：for 循环变量**在循环体结束后更新**。
- 黄金法则：避免在循环体内非常规修改循环变量。
- 程序模板：

```
for(int i=0; i<n; ) {
    if(skip_cond) {
        i += 2; // 非常规更新
    } else {...}
    i++; // 常规更新
}
```

(三) 常见题型

1、选择结构

(1) 闰年判断

规则：能被 4 整除但不能被 100 整除，或能被 400 整除

```
if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {  
    cout << "Leap year";  
}
```

(2) 三角形类型判断

规则：先判断能否构成三角形，再分类

```
// 1. 判断能否构成三角形  
if (a+b>c && a+c>b && b+c>a) {  
    // 2. 判断具体类型  
    if (a==b && b==c)  
        cout << "Equilateral"; // 等边  
    else if (a==b || b==c || a==c)  
        cout << "Isosceles"; // 等腰  
    else if (a*a+b*b==c*c || a*a+c*c==b*b || b*b+c*c==a*a)  
        cout << "Right-angled"; // 直角  
    else  
        cout << "Scalene"; // 普通  
} else {  
    cout << "Not a triangle";  
}
```

(3) 奇偶性判断

// 方法 1: 取模法

```
if (num % 2 == 0)
```

```
    cout << "Even";
```

// 方法 2: 位运算 (高效)

```
if ((num & 1) == 0)
```

```
    cout << "Even";
```

- 偶数: 如果一个数是偶数, 那么它的二进制最后一位是 0。
- 奇数: 如果一个数是奇数, 那么它的二进制最后一位是 1。
- $num \& 1$: 如果是 1, 说明是奇数; 如果是 0, 说明是偶数。

运算符	描述	运算规则
&	与	两个位都为 1 时, 结果才为 1

二进制	十进制
0000	0
0001	1
0010	2
0011	3
0100	4

(4) 数位末尾判断

// 判断末位是否为 0

```
if (num % 10 == 0)
```

```
    cout << "Ends with 0";
```

// 判断末位是否为 5

```
if (num % 10 == 5)
```

```
    cout << "Ends with 5";
```

// 判断末两位是否为 25

```
if (num % 100 == 25)
```

```
    cout << "Ends with 25";
```

(5) 质数判断

规则: 只能被 1 和自身整除的数是质数 (素数)。

```
bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i=2; i*i<=n; i++)
        if (n % i == 0)
            return false;
    return true; }
```

Prime Numbers									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

(6) 水仙花数判断

规则：三位数且各位立方和等于自身

```

int a = num/100;      // 百位
int b = num/10 % 10;  // 十位
int c = num % 10;     // 个位

if (a*a*a + b*b*b + c*c*c == num)

    cout << "Narcissistic";

```

(7) 回文数判断

规则：正序和倒序相同

```

int temp = num, rev = 0;
while (temp > 0) {
    rev = rev*10 + temp%10;
    temp /= 10;
}

if (num == rev)

    cout << "Palindrome";

```

Palindrome

1534351

(8) 完全数判断

规则：所有真因子之和等于自身

```

int sum = 0;
for (int i=1; i<=num/2; i++)
    if (num % i == 0)
        sum += i;

if (sum == num)

    cout << "Perfect number";

```

							$6 = 1 \times 6$
							$6 = 2 \times 3$
							$6 = 3 \times 2$
							$6 = 6 \times 1$
							$1 + 2 + 3$
							$= 6$

(9) 日期合法性判断

```
if (year < 1 || month < 1 || month > 12 || day < 1)
    return false;

// 各月份天数验证
int daysInMonth[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};

if ((year%4==0 && year%100!=0) || year%400==0)
    daysInMonth[2] = 29; // 闰年 2 月 29 天

if (day > daysInMonth[month])
```

(10) 成绩等级判断

```
if (score >= 90)
    grade = 'A';
else if (score >= 80)
    grade = 'B';
else if (score >= 70)
    grade = 'C';
else if (score >= 60)
    grade = 'D';
else
    grade = 'F';
```

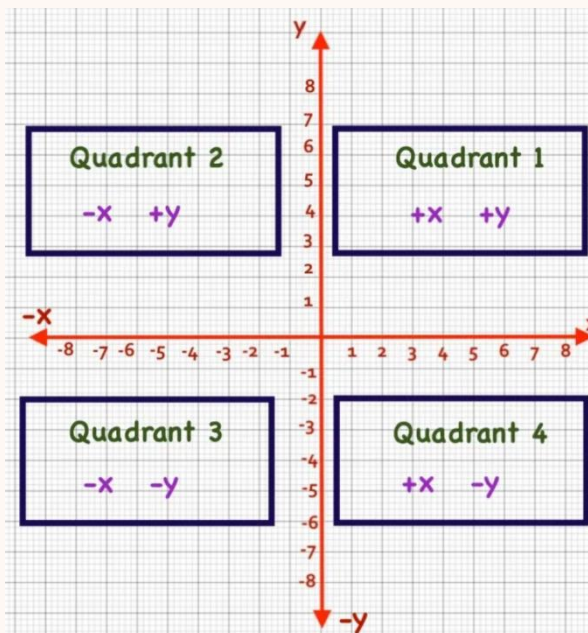
(11) 坐标象限判断

```
if (x > 0 && y > 0)
    cout << "Quadrant I";

else if (x < 0 && y > 0)
    cout << "Quadrant II";
```

```

else if (x < 0 && y < 0)
    cout << "Quadrant III";
else if (x > 0 && y < 0)
    cout << "Quadrant IV";
else if (x==0 && y!=0)
    cout << "Y-axis";
else if (y==0 && x!=0)
    cout << "X-axis";
else
    cout << "Origin";
    
```



(12) 字符类型判断

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	B
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	C
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	D
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	E
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL	F

```

if (ch >= '0' && ch <= '9')
    cout << "Digit";
else if (ch >= 'A' && ch <= 'Z')
    cout << "Uppercase";
else if (ch >= 'a' && ch <= 'z')
    cout << "Lowercase";
else
    
```

```
cout << "Special character";
```

2、循环结构

(1) 区间求和模板

```
// 2011 年真题：计算 [n, m] 内整数和

#include <iostream>

using namespace std;

int main() {

    int n, m, sum = 0;

    cin >> n >> m;

    for(int i = n; i <= m; i++) // 包含端点

        sum += i;

    cout << sum;

    return 0;

}
```

变式训练：

- 计算 1~100 偶数和：if(i%2==0) sum+=i;
- 计算 100 以内被 5 整除余 1 的和：if(i%5==1) sum+=i;

(2) 因子统计模板

```
// 2012 年真题：计算 n 的因子个数

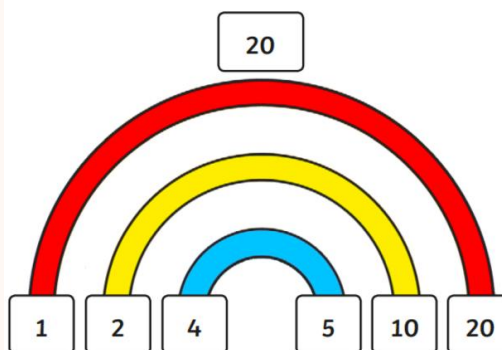
#include <iostream>

using namespace std;

int main() {

    int n, cnt = 0;

    cin >> n;
```



```

for(int i = 1; i <= n; i++)
    if(n % i == 0) // i 是因子
        cnt++;

cout << cnt;

return 0;
}

```

优化:

```

// 优化到 O(sqrt(n))
for(int i = 1; i*i <= n; i++) {
    if(n % i == 0) {
        cnt++;
        if(i != n/i) cnt++; // 避免重复计数
    }
}

```

1	×	36	=	36
2	×	18	=	36
3	×	12	=	36
4	×	9	=	36
6	×	6	=	36

(3) 倍数统计模板

// 2013 年真题: 统计 $[a, b]$ 内 k 的倍数

```

#include <iostream>
using namespace std;

int main() {
    int a, b, k, cnt = 0;
    cin >> a >> b >> k;
    for(int i = a; i <= b; i++)
        if(i % k == 0) // 整除判断
            cnt++;
}

```

6	×	0	=	0
6	×	1	=	6
6	×	2	=	12
6	×	3	=	18
6	×	4	=	24
6	×	5	=	30
6	×	6	=	36
6	×	7	=	42
6	×	8	=	48
6	×	9	=	54
6	×	10	=	60

A few Multiples of 6

```

    cout << cnt;

    return 0;

}

```

变式:

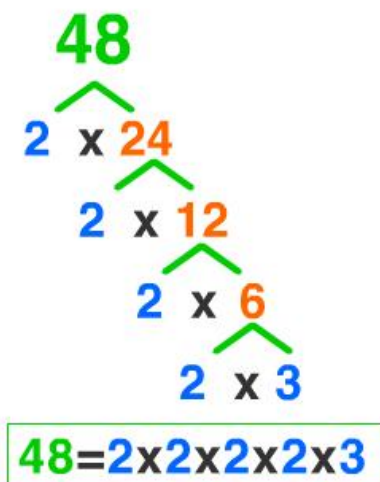
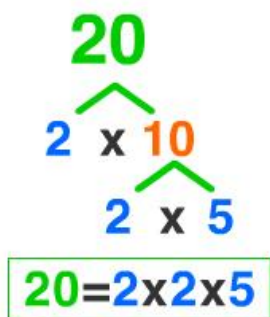
- 被 3 整除余 2: `if(i%3 == 2)`
- 同时被 2 和 3 整除: `if(i%6 == 0)`

(4) 质因数分解模板

```

// 2020 年真题：分解质因数
#include <iostream>
using namespace std;
int main() {
    int n;
    cin >> n;
    for(int i = 2; i*i <= n; i++) {
        while(n % i == 0) {
            cout << i << " ";
            n /= i;
        }
    }
    if(n > 1) cout << n; // 剩余的大质数
    return 0;
}

```

**(6) 二分查找模板**

```

// 2013 年真题：二分查找

```

```
#include <iostream>
```

```
using namespace std;
```

```
const int N = 1005;
```

```
int a[N];
```

```
int main() {
```

```
    int n, target;
```

```
    cin >> n >> target;
```

```
    for(int i = 0; i < n; i++) cin >> a[i];
```

```
    int left = 0, right = n-1;
```

```
    while(left <= right) {
```

```
        int mid = (left + right) / 2;
```

```
        if(a[mid] == target) {
```

```
            cout << mid + 1; // 位置从 1 计数
```

```
            return 0;
```

```
        }
```

```
        if(target < a[mid]) right = mid - 1;
```

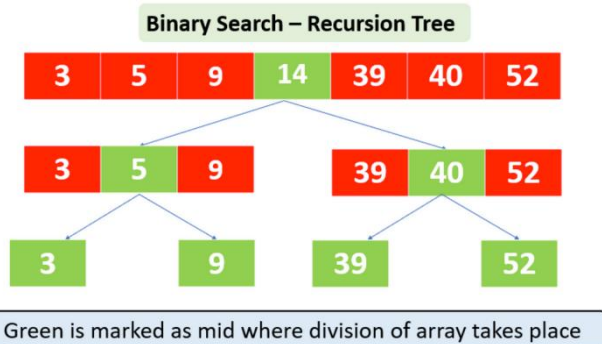
```
        else left = mid + 1;
```

```
    }
```

```
    cout << "Not found";
```

```
    return 0;
```

```
}
```



(7) 快速幂模板 (了解)

快速幂就是快速算底数的 n 次幂。当 n 巨大无比时候，可能超出计算机运算范围。

如果让我们算 $(2^{100000000000})\%10000$ 。快速幂时间复杂度为 $O(\log_2 n)$ ，与朴素的 $O(n)$ 相比效率有了极大的提高。

我似乎发现了什么东西

$$(a * b) \% p = (a \% p * b \% p) \% p$$

emmmmm...



$$\begin{aligned} 2^n &= (2 * 2 \dots * 2) \text{ (共 } n \text{ 个相乘)} \\ &= 2 * (2 * 2 \dots * 2) \text{ (} n-1 \text{ 个)} \\ &= \dots \\ &= (2 * 2 \dots * 2) * (2 * 2 \dots * 2) \text{ (各一半)} \\ &= 2^{(n/2)} * 2^{(n/2)} \end{aligned}$$

懂了懂了，原来不难嘛

如果b是偶数，那么 $a^b = a^{2 * (b/2)} = (a^2)^{b/2}$

如果b是奇数，那么 $a^b = a * a^{2 * (b/2)} = a * (a^2)^{b/2}$

emmmmm...



所以那偶数来看带上取模的公式，具体计算是这样的：

$$a^b \% p = (a^2)^{b/2} \% p = ((a \% p) * (a \% p))^{b/2} \% p$$

再下一次就是新的一次计算，一直到幂的次数为0或1就ok的

你废了吗？

快速幂真的太牛了，是个logn级别的时间复杂度。
有多牛给你举个栗子：

$$2^{100000} \% 1e7 = (2^2)^{50000} \% 1e7$$

$$\text{第一次: } ((2 \% 1e7) * (2 \% 1e7))^{50000} \% 1e7 = 4^{50000} \% 1e7$$

$$\text{第二次: } ((4 \% 1e7) * (4 \% 1e7))^{25000} \% 1e7 = 16^{25000} \% 1e7$$

emmmmm...



再下一次就是新的一次计算，一直到幂的次数为0或1就ok的

如果刚开始次数为n，我用for循环要执行n次，但是这个快速幂是logn次。这个数据越大越明显，你要知道log级别执行32次就达到int最大值，64次就达到long的最大值哦

```
// 2017 年真题：快速幂取模

#include <iostream>

using namespace std;

typedef long long LL;

LL fastPow(LL x, LL p, LL m) {
    LL res = 1;
    while(p) {
        if(p & 1) res = res * x % m; // p 为奇数
        x = x * x % m;
        p >>= 1; // p /= 2
    }
    return res;
}

int main() {
    LL x, p, m;
    cin >> x >> p >> m;
    cout << fastPow(x, p, m);
    return 0;
}
```

(8) 环检测模板 (了解)

计算一个排列 (permutation) 中包含多少个“环 (cycle)”。

程序功能:

输入一个长度为 n 的排列 (数组 d) ,

输出这个排列中包含多少个置换环 (Cycle) 。

知识点解释

①排列 (Permutation)

排列是一个包含了 n 个不同整数 (0 到 $n-1$) 的数组, 其中每个数字只出现一次。

比如:

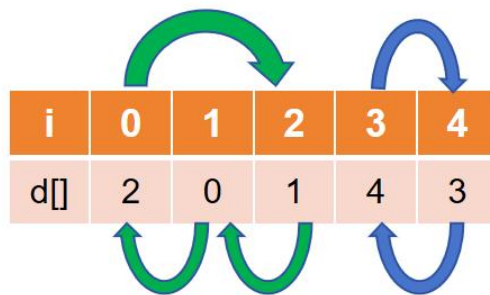
- 输入 $n = 5$, 数组为: $[2, 0, 1, 4, 3]$
- 这是一个排列。我们可以把它看作一个映射关系, 即每个 i 被映射到 $d[i]$ 。

②置换环 (Cycle)

每个数通过排列跳来跳去, 最终会回到起点, 形成一个环。

比如上面例子 $[2, 0, 1, 4, 3]$:

- 从 0 开始: $0 \rightarrow 2 \rightarrow 1 \rightarrow 0$ 是一个环。
- 从 3 开始: $3 \rightarrow 4 \rightarrow 3$ 是另一个环。
- 所以有 2 个环。



③数组和访问标记 (Visited 标记)

程序使用 `bool vis[N]` 来记录哪些位置已经访问过, 避免重复统计。

```
#include <cstring>
```

引入输入输出库和字符串操作库 (主要为了 `memset`) 。

```
const int N = 105;

int d[N];          // 存放排列

bool vis[N];       // 标记每个位置是否已经访问过

int main() {

    int n, cnt = 0;

    cin >> n;                          // 输入排列长度

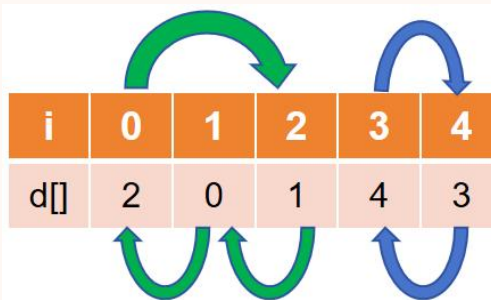
    for(int i = 0; i < n; i++) cin >> d[i]; // 输入排列内容
```

输入排列, 如: $n=5$, 接着输入: $2\ 0\ 1\ 4\ 3$

```
memset(vis, 0, sizeof(vis)); // 初始化所有位置都没访问过
```

memset 是 C++ 中的一个函数, 用来把内存清零, 这里清空了 vis 数组。

```
for(int i = 0; i < n; i++) {
    if(!vis[i]) {
        for(int j = i; !vis[j]; j = d[j])
            vis[j] = true;
        cnt++;
    }
}
```



- 遍历每一个下标 i。
- 如果 i 没有访问过, 就从 i 出发, 沿着 $d[i] \rightarrow d[d[i]] \rightarrow \dots$ 一直跳, 直到回到原点, 形成一个环。
- 每次这样遍历一个环, 就给 cnt++。

举例解释:

- 如果排列是 [2, 0, 1, 4, 3], 我们从 0 开始:
 - $0 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow$ 一个环, 标记 vis[0]、vis[2]、vis[1]
- 然后从 3 开始:
 - $3 \rightarrow 4 \rightarrow 3 \rightarrow$ 另一个环
- 最终, cnt == 2。

输出环的数量。

完整版程序

```
// 2018 年真题: 计算排列中的环数量
#include <iostream>
#include <cstring>
using namespace std;
const int N = 105;
```

```

int d[N];

bool vis[N];

int main() {
    int n, cnt = 0;

    cin >> n;

    for(int i = 0; i < n; i++) cin >> d[i];

    memset(vis, 0, sizeof(vis));

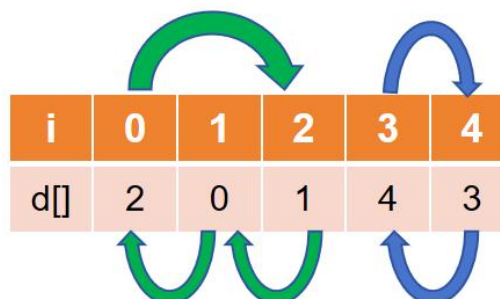
    for(int i = 0; i < n; i++) {
        if(!vis[i]) {
            for(int j = i; !vis[j]; j = d[j])
                vis[j] = true;

            cnt++;
        }
    }

    cout << cnt;

    return 0;
}

```



(四) 做题方法总结

1、顺序结构

- 理解每一行代码的作用：从 **main 函数开始逐行分析** 程序中的赋值、运算和输入输出。
- 手动追踪变量的值：在纸上模拟程序，**逐步计算每个变量的变化**。
- 输出结果：根据代码的执行顺序确定最终的输出。

2、选择结构

(1) 做题方法

- 理解条件判断：仔细分析每个**条件**的意义，以及程序在不同条件下的执行路径。
- 分步模拟判断：如果条件有多个，**手动模拟每个判断**，看看程序会执行哪一条路径。
- 输出的不同情况：根据条件判断的结果确定程序的输出。

(2) 做题技巧

- 明确条件的判断逻辑：理解每个 if 或 switch 的判断条件，并考虑每个条件**成立**与**不成立**的情况。
- 检查“嵌套”条件：有时条件判断是嵌套的，需要**逐层分析**条件。
- 模拟程序的不同分支：尤其在 if-else 或 switch 中，根据不同的输入模拟程序的不同分支。

2、循环结构

(1) 做题方法

做题时，要关注以下几点：

- **循环次数**：理解循环的终止条件，判断循环会执行多少次。
- 循环**变量的变化**：注意循环控制变量的变化方式，计算每次循环后变量的值。
- 循环体的执行逻辑：注意每次循环中执行的语句，是否存在改变程序状态或变量的操作。

(2) 做题技巧

- 模拟循环：手动计算每次循环的变量值，尤其在多层嵌套循环时。
- 确认**循环条件**：弄清楚循环的条件是如何控制循环执行的，尤其要关注循环的退出条件。
- 考虑循环中间的修改：如果循环内有条件修改变量（例如 break），要特别注意这些修改对结果的影响。

(五) 课后练习

第 1 题. 2014 年第 23 题 (顺序结构)

```
#include <iostream>

using namespace std;
```

```
int main()
{
    int a, b, c, d, ans;

    cin >> a >> b >> c;

    d = a - b;

    a = d + c;

    ans = a * b;

    cout << "Ans = " << ans << endl;

    return(0);
}
```

输入: 2 3 4

输出: Ans = ____

答案: 9

```
2 3 4
Ans = 9
```

详细解析:

1. 初始赋值:

-a = 2 (来自输入的第一个值)

-b = 3 (来自输入的第二个值)

-c = 4 (来自输入的第三个值)

2. 计算步骤:

-d = a - b = 2 - 3 = -1 (变量 d 存储中间结果)

-a = d + c = -1 + 4 = 3 (关键覆盖: 变量 a 被更新为 3, 原始值 2 丢失)

-ans = a * b = 3 * 3 = 9 (使用更新后的 a 和原始 b 计算)

3. 输出结果:

- 输出字符串 "Ans = " 后接 ans 的值 9

易错点总结:

- 变量覆盖陷阱：a 在第二步被重新赋值（从 2→3），后续计算使用新值
- 整数运算：所有运算均为整数运算，无浮点转换问题
- 数据流向：原始输入值 c 仅用于更新 a，不直接参与最终计算

第 2 题. 2015 年第 23 题 (选择结构)

```
#include <iostream>

using namespace std;

int main()
{
    int a, b, c; a = 1;

    b = 2;

    c = 3;

    if(a > b)
        if(a > c)
            cout << a << ' ';

        else
            cout << b << ' ';

    cout << c << endl;

    return 0;

}
```

输出： ____

答案： 3

详细解析：

3

1. 初始赋值：

-a = 1

-b = 2

-c = 3

2. 选择结构执行流程:

- 外层 if(a > b): 1 > 2 为假 → 跳过整个内层 if-else 块
- 内层 if-else 未被触发 (包括 cout << a << ' '; 和 cout << b << ' ');
- 执行最后的独立语句: cout << c << endl; → 输出 3

3. 缩进误导分析:

- 尽管 else 缩进与第一个 if 对齐, 但 C++ 中 else 始终匹配最近未匹配的 if
- 实际匹配关系:

```

if(a > b)           // 外层 if (条件为假)
    if(a > c)       // 内层 if (未执行)
        ...
    else            // 此 else 属于内层 if (未执行)
        ...

```

黄金法则应用:

- else 匹配原则: 当存在嵌套 if 时, **else 总是匹配最近的 if**
- 作用域明确: 建议用 {} 明确作用域避免混淆:

```

if(a > b) {
    if(a > c) { ... }
} else { ... } // 明确匹配外层 if

```

第 3 题. 2010 年第 24 题 (循环结构 while)

```

#include <iostream>

using namespace std;

int rSum(int j)
{

```

```

int sum = 0;

while (j != 0) {
    sum = sum * 10 + (j % 10);
    j = j / 10;
}

return sum;
}

int main()
{
    int n, m, i;
    cin >> n >> m;

    for (i = n; i < m; i++)
        if (i == rSum(i))
            cout << i << ' ';

    return 0;
}

```

输入: 90 120

输出: ____

答案: 99 101 111

```

90 120
99 101 111

```

详细解析:

1. 函数 rSum 功能:

- 反转数字 (如输入 123 → 输出 321)
- 执行过程:

```

while(j != 0) {
    sum = sum*10 + j%10; // 取末位拼接成反转数
}

```

```

        j = j/10;           // 移除末位
    }

```

2. 主程序逻辑:

- 遍历区间[n, m) (左闭右开) : i 从 90 到 119
- 检查条件: $i == rSum(i)$ (回文数判断)

3. 回文数验证:

i	rSum(i)	是否相等	原因
90	9	否	$90 \rightarrow 09 = 9$
91	19	否	
99	99	是	99 反转仍为 99
100	1	否	$100 \rightarrow 001 = 1$
101	101	是	101 反转仍为 101
111	111	是	11 反转仍为 111
120	21	否	$120 \rightarrow 021 = 21$

关键结论:

- 回文数条件: 数字反转后与原数相等
- 特例排除: 100 反转后为 1 (末位 0 被丢弃)

第 4 题. 2011 年第 23 题 (循环结构 while)

```

#include<iostream>

using namespace std;

int main()
{
    int i,n,m,ans;

```

```

cin>>n>>m;

i=n;

ans=0;

while(i<=m){

    ans+=i;

    i++;

}

cout<<ans<<endl;

return 0;

}

```

输入: 10 20

输出: ____

答案: 165

```

10 20
165

```

详细解析:

1. 程序功能: 计算区间 $[n, m]$ 内所有整数的和 (含端点)

2. 执行流程:

- 初始化: $i = n = 10, ans = 0$
- 循环条件: $i \leq m$ (即 $i \leq 20$)
- 循环体:
 - $ans += i$ (累加当前 i)
 - $i++$ (i 自增 1)

3. 累加过程:

正确计算:

- 从 10 到 20 共 11 个数
- 等差数列求和公式:

$$S = (\text{首项} + \text{尾项}) \times \text{项数} / 2$$

$$= (10 + 20) \times 11 / 2$$

$$= 30 \times 11 / 2$$

$$= 165$$

- 验证: $10+11+12+13+14+15+16+17+18+19+20 = 165$

易错点:

- 端点包含: 循环条件 $i \leq m$ 包含右端点 20
- 项数计算: 项数 $= m - n + 1 = 20 - 10 + 1 = 11$

第 5 题. 2018 年第 19 题 (循环结构 for)

```
#include <stdio.h>

int main() {
    int x;

    scanf("%d", &x);

    int res = 0;

    for (int i = 0; i < x; ++i) {
        if (i * i % x == 1) {
            ++res;
        }
    }

    printf("%d", res);

    return 0;
}
```

输入: 15

输出: ____

15
4

答案: 4

详细解析：

1. 程序功能：统计满足 $i^2 \% x == 1$ 的整数 i 的个数 ($i \in [0, x-1]$)

2. 枚举验证 ($x=15$)：

i	i^2	$i^2 \% 15$	是否满足条件
0	0	0	否
1	1	1	是
2	4	4	否
3	9	9	否
4	16	1	是
5	25	10	否
6	36	6	否
7	49	4	否
8	64	4	否
9	81	6	否
10	100	10	否
11	121	1	是
12	144	9	否
13	169	4	否
14	196	1	是

3. 满足条件的 i : 1, 4, 11, 14 (共 4 个)

数学扩展：

- 当 x 为质数时，解的数量为 2 (1 和 $x-1$)
- 当 x 为合数时，解的数量可能增加 (如本例 $x=15$ 有 4 解)

第 6 题

在 C++ 中，表达式 $5 / 2$ 的结果是？

A) 2.5

B) 2

C) 3

D) 2.0

答案： B

2

解析：

在 C++ 中，当两个整数进行除法运算时，结果也是整数（即向下取整）。 $5/2$ 等于 2.5，但由于操作数都是整数，结果会被截断为整数 2。因此，选项 B 正确。选项 A 和 D 涉及浮点数，但此处未使用浮点类型；选项 C 是向上取整，不符合整数除法规则。

第 7 题 【2012 年第 23 题】

输入：1 2 5，以下程序的输出结果是？

```
#include <iostream>

using namespace std;

int a, b, c, d, e, ans;

int main()
{
    cin >> a >> b >> c;

    d = a + b;

    e = b + c;

    ans = d + e;

    cout << ans << endl;

    return 0;
}
```

A) 3

B) 7

C) 10

D) 编译错误

答案: C

```
1 2 5
10
```

解析:

程序读取三个整数 a 、 b 、 c , 计算 $d = a + b$ 和 $e = b + c$, 然后 $ans = d + e$ 。

输入为 1 2 5 时:

$$d = 1 + 2 = 3$$

$$e = 2 + 5 = 7$$

$$ans = 3 + 7 = 10$$

因此输出为 10。程序无语法错误, 不会编译错误, 选项 C 正确。

第 8 题 【2013 年第 23 题】

输入: 3 5, 以下程序的输出结果是?

```
#include <iostream>

using namespace std;

int main()
{
    int a, b;

    cin >> a >> b;

    cout << a << "+" << b << "=" << a + b << endl;

}
```

A) 3

B) 5

C) 8

D) 3+5=8

答案: D

```
3 5
3+5=8
```

解析:

程序读取两个整数 a 和 b ，并输出格式为 $a + b =$ 和 的字符串。输入 3 5 时：

$a = 3, b = 5, a + b = 8$

输出字符串为 $3+5=8$

因此选项 D 正确。选项 A 和 B 是输入值，选项 C 是计算结果，但输出包含完整格式字符串。

第 9 题 【2012 年第 24 题】

输入：18，以下程序的输出结果是？

```
#include <iostream>

using namespace std;

int n, i, ans;

int main()
{
    cin >> n;
    ans = 0;
    for (i = 1; i <= n; i++)
        if (n % i == 0) ans++;
    cout << ans << endl;
    return 0;
}
```

- A) 4
- B) 5
- C) 6
- D) 7

答案： C

18
6

解析：

程序计算输入整数 n 的因子个数（即能整除 n 的正整数个数）。输入 18 时，因子有：1, 2, 3, 6, 9, 18，共 6 个。因此输出为 6。选项 C 正确。其他选项是常见错误，如遗漏因子或计数错误。

第 10 题 【2013 年第 19 题】

下列程序中，正确计算 1,2,...,100 这 100 个自然数之和 sum（初始值为 0）的是（ ）。

A. `i = 1; do{ sum +=i; i++; }while(i<=100);`

B. `i = 1; do{ sum +=i; i++; }while(i > 100);`

C. `i = 1; while(i < 100){ sum+=i; i++; }`

D. `i = 1; while(i >= 100){ sum+=i; i++; }`

答案： A

解析：

选项 A 使用 do-while 循环，从 $i=1$ 开始累加到 sum，直到 $i \leq 100$ ，循环体至少执行一次，正确覆盖 1 到 100。

选项 B：条件 $i > 100$ 初始为假，循环不执行，sum 保持 0。

选项 C：条件 $i < 100$ ，当 $i=100$ 时不执行，只累加 1 到 99。

选项 D：条件 $i \geq 100$ 初始为假，循环不执行。

因此只有 A 正确。

第 11 题 【2013 年第 24 题】

以下程序的功能是计算区间内能被整除的数的个数。输入： 1 100 15，程序运行后输出的结果是？

```
#include <iostream>

using namespace std;

int main()
{
    int a, b, u, i, num;

    cin >> a >> b >> u;
```

```

    num = 0;

    for (i = a; i <= b; i++)

        if ((i % u) == 0)

            num++;

    cout << num << endl;

    return 0;

}

```

- A) 4
- B) 5
- C) 6
- D) 7

答案： C

```

1 100 15
6

```

解析：

程序计算从 a 到 b 之间能被 u 整除的整数个数。输入 1 100 15 时，范围为 1 到 100，求能被 15 整除的数：15, 30, 45, 60, 75, 90，共 6 个。因此输出为 6。选项 C 正确。其他选项是常见错误，如范围或整除条件理解错误。

第 12 题 【2014 年第 19 题】

若有如下程序段，其中 s, a, b, c 均已定义为整型变量，且 a, c 均已赋值， $c > 0$ 。

```

s = a;

for(b = 1; b <= c; b++)

    s += 1;

```

则与上述程序段功能等价的赋值语句是()。

- A. $s = a + b$
- B. $s = a + c$
- C. $s = s + c$

D. $s = b + c$

答案： B

解析：

程序段中, s 初始化为 a , 然后循环 c 次, 每次 s 增加 1, 相当于 s 最终值为 $a + c$ 。

选项 A: b 是循环变量, 最终值为 $c+1$, 但未直接使用。

选项 C: $s = s + c$ 会依赖 s 的初始值, 但原程序是累加 1, 不是直接加 c 。

选项 D: 与 a 无关。

因此选项 B $s = a + c$ 等价。

第 13 题 【2019 年第 4 题】

若有如下程序段, 其中 s 、 a 、 b 、 c 均已定义为整型变量, 且 a 、 c 均已赋值 (c 大于 0)

```
s = a;
for (b = 1; b <= c; b++) s = s - 1;
```

则与上述程序段功能等价的赋值语句是 ()

A. $s = a - c$;

B. $s = a - b$;

C. $s = s - c$;

D. $s = b - c$;

答案： A

解析：

s 初始为 a , 循环 c 次, 每次 s 减 1, 最终 $s = a - c$ 。选项 A 正确。

选项 B: b 是循环变量, 最终为 $c+1$, 但未在赋值中使用。

选项 C: $s = s - c$ 依赖于 s 的当前值, 但原程序是逐步减少。

选项 D: 与 a 无关。

功能等价于 $s = a - c$ 。

第 14 题 【2014 年第 13 题】

要求以下程序的功能是计算： $s = 1 + 1/2 + 1/3 + \dots + 1/10$ 。程序运行后输出结果错误，导致错误结果的程序行是()。

```
#include <iostream>

using namespace std;

int main()
{
    int n;

    float s;

    s = 1.0;

    for(n = 10; n > 1; n--)

        s = s + 1 / n;

    cout << s << endl;

    return 0;

}
```

- A. $s = 1.0;$
- B. $\text{for}(n = 10; n > 1; n--)$
- C. $s = s + 1 / n;$
- D. $\text{cout} << s << \text{endl};$

答案： C

解析：

错误发生在行 $s = s + 1 / n;$ 。 $1 / n$ 中， n 是整数，因此进行整数除法：当 $n > 1$ 时， $1/n$ 结果为 0（例如 $1/2=0$ ）。正确做法应使用浮点除法，如 $1.0 / n$ 。其他行无错误：A 行初始化 s 正确；B 行循环范围正确（从 10 到 2）；D 行输出正常。整数除法导致结果错误，选项 C 正确。

第 15 题 【2016 年第 13 题】

有以下程序：

```
#include <iostream>

using namespace std;

int main()
{
    int k = 4, n = 0;
    while (n < k)
    {
        n++;
        if (n % 3 != 0)
            continue;
        k--;
    }
    cout << k << "," << n << endl;
    return 0;
}
```

程序运行后输出的结果是（ ）。

- A. 2,2
- B. 2,3
- C. 3,2
- D. 3,3

答案： D

3,3

解析：

循环条件 $n < k$ ，初始 $k=4, n=0$ 。逐步执行：

循环 1: $n=1$ ($n\%3=1\neq 0$ ，执行 `continue`，跳过 `k--`)

循环 2: $n=2$ ($n\%3=2\neq0$, 执行 `continue`, 跳过 `k--`)

循环 3: $n=3$ ($n\%3=0$, 不执行 `continue`, 执行 `k--`, $k=3$)

此时 $n=3, k=3$, 条件 $n < k$ 为假, 循环结束。输出 $k=3, n=3$ 。选项 D 正确。

第 16 题 【2014 年第 15 题】

有以下程序:

```
#include <iostream>

using namespace std;

int main()
{
    int s, a, n;

    s = 0;

    a = 1;

    cin >> n;

    do
    {
        s += 1;

        a -= 2;

    }

    while ( a != n );

    cout << s << endl;

    return(0);
}
```

若要使程序的输出值为 2, 则应该从键盘给 n 输入的值是()。

A. -1

B. -3

C. -5

D. 0

答案: B

解析:

-3
2

程序初始 $a=1, s=0$ 。循环中每次 s 加 1 (计数), a 减 2, 直到 $a == n$ 时停止。输出 $s=2$ 表示循环执行了 2 次:

第一次: $s=1, a=1-2=-1$ 第二次: $s=2, a=-1-2=-3$

循环条件 $a != n$, 当 $n=-3$ 时, 第二次后 $a=-3$, 等于 n , 循环停止, 输出 $s=2$ 。因此输入 $n=-3$, 选项 B 正确。

第 17 题 【2010 年第 23 题】

第一行输入: 91 2 20, 第二行输入 77, 以下程序的输出结果是?

```
#include <iostream>

using namespace std;

void swap(int & a, int & b)
{
    int t;

    t = a;
    a = b;
    b = t;
}

int main()
{
    int a1, a2, a3, x;

    cin >> a1 >> a2 >> a3;
```

```
    if (a1 > a2)
        swap(a1, a2);
    if (a2 > a3)
        swap(a2, a3);
    if (a1 > a2)
        swap(a1, a2);

    cin >> x;
    if (x < a2)
        if (x < a1)
            cout << x << ' ' << a1 << ' ' << a2 << ' ' << a3 << endl;
        else
            cout << a1 << ' ' << x << ' ' << a2 << ' ' << a3 << endl;
    else
        if (x < a3)
            cout << a1 << ' ' << a2 << ' ' << x << ' ' << a3 << endl;
        else
            cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << x << endl;

    return 0;
}
```

A) 2 20 77 91

B) 2 77 20 91

C) 77 2 20 91

D) 2 20 91 77

答案： A

解析:

程序先对输入的三个数排序, 然后根据第四个数 x 插入到已排序序列中。

第一行输入 91 2 20: 经过三个 if 和 swap 后, 排序为 $a1=2, a2=20, a3=91$ 。

第二行输入 $x=77$:

```
91 2 20
77
2 20 77 91
```

比较 $x < a2$ ($77 < 20$? 假), 进入 else 分支。

比较 $x < a3$ ($77 < 91$? 真), 输出 $a1\ a2\ x\ a3$, 即 2 20 77 91。

因此选项 A 正确。其他选项错误, 如 B 和 C 顺序混乱, D 未正确插入。

第 18 题 【2007 年第 13 题】

一个无法靠自身的控制终止的循环称为“死循环”, 例如, 在 C++ 语言程序中, 语句 `while(1) printf("*");` 就是一个死循环, 运行时它将无休止地打印 * 号。下面关于死循环的说法中, 只有 () 是正确的。

A. 不存在一种算法, 对任何一个程序及相应的输入数据, 都可以判断是否会出现死循环, 因而, 任何编译系统都不做死循环检查

B. 有些编译系统可以检测出死循环

C. 死循环属于语法错误, 既然编译系统能检查各种语法错误, 当然也应该能检查出死循环

D. 死循环与多进程中出现的“死锁”差不多, 而死锁是可以检测的, 因而, 死循环也可以检测的

答案: A

解析:

死循环是运行时行为, 不属于语法错误 (选项 C 错误)。根据计算理论 (停机问题), 不存在通用算法能检测所有程序的死循环 (选项 A 正确)。编译系统通常检测语法错误, 但无法检测所有死循环 (选项 B 错误)。死锁是多进程资源竞争问题, 可检测, 但死循环是单进程控制流问题, 与之不同 (选项 D 错误)。因此 A 是正确说法。

第 19 题 【2020 年第 19 题】 (质因数分解)

(质因数分解) 给出正整数 n , 请输出将 n 质因数分解的结果, 结果从小到大输出。

例如: 输入 $n=120$, 程序应该输出 2 2 2 3 5, 表示: $120=2 \times 2 \times 2 \times 3 \times 5$ 。输入保证 $2 \leq n \leq 10^9$ 。

提示：先从小到大枚举变量 i ，然后用 i 不停试除 n 来寻找所有的质因子。

试补全程序。

```
#include <cstdio>

using namespace std;

int n, i;

int main() {
    scanf("%d", &n);
    for (i = 【①】; 【②】 <= n; i++) {    //
        【③】 {                          //
            printf("%d ", i);
            n = n / i;                    //
        }
    }

    if (【④】)                            //
        printf("%d ", 【⑤】);           //

    return 0;

}
```

选择题：

①处应填 ()

- A. 1
- B. $n-1$
- C. 2
- D. 0

答案： C

解析： 质因数分解需从最小质数 2 开始枚举，因此 i 初始值为 2。

②处应填 ()

- A. n/i
- B. $n/(i*i)$
- C. $i*i$
- D. $i*i*i$

答案: C

解析: 循环条件 $i*i \leq n$ 确保只枚举到 \sqrt{n} , 避免无效枚举。

③处应填 ()

- A. `if(n%i==0)`
- B. `if(i*i<=n)`
- C. `while(n%i==0)`
- D. `while(i*i<=n)`

答案: C

解析: `while` 循环用于连续用相同质因数试除 (如 120 需连续除 3 次 2)。

④处应填 ()

- A. $n > 1$
- B. $n \leq 1$
- C. $i < n/i$
- D. $i + i \leq n$

答案: A

解析: 循环结束后, 若 $n > 1$ 说明剩余 n 是质因数 (如 7 分解后剩余 7)。

⑤处应填 ()

- A. 2
- B. n/i
- C. n
- D. i

答案： C

解析： 直接输出剩余的质因数 n（如输入 11 时输出 11）。

完整版程序

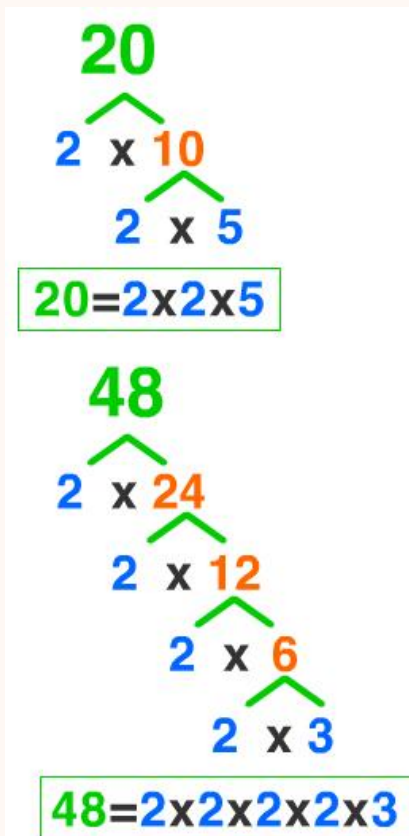
```
#include <cstdio>

using namespace std;

int n, i;

int main() {
    scanf("%d", &n);
    for(i = 2; i * i <= n; i++){
        while(n % i == 0){
            printf("%d ", i);
            n = n / i;
        }
    }
    if(n > 1)
        printf("%d ", n);
    return 0;
}
```

```
120
2 2 2 3 5
```



第 20 题【2022 年第 19 题】（枚举因数）

```
#include <bits/stdc++.h>

using namespace std;

int main() {

    int n;

    cin >> n;

    vector<int> fac;
```

```

    fac.reserve((int)ceil(sqrt(n)));

    int i;

    for (i = 1; i * i < n; ++i) {
        if (【①】) { //
            fac.push_back(i);
        }
    }

    for (int k = 0; k < fac.size(); ++k) {
        cout << 【②】 << " "; //
    }

    if 【③】) { //
        cout << 【④】 << " "; //
    }

    for (int k = fac.size() - 1; k >= 0; --k) {
        cout << ⑤ << " "; //
    }
}

```

选择题：

①处应填 ()

- A. $n \% i == 0$
- B. $n \% i == 1$
- C. $n \% (i-1) == 0$
- D. $n \% (i-1) == 1$

答案： A

解析： 判断 i 是否为 n 的因数需满足 $n \% i == 0$ 。

②处应填 ()

- A. $n / \text{fac}[k]$
- B. $\text{fac}[k]$
- C. $\text{fac}[k]-1$
- D. $n / (\text{fac}[k]-1)$

答案: B

解析: 第一轮输出小于 \sqrt{n} 的因数, 直接输出 $\text{fac}[k]$ 。

③处应填 ()

- A. $(i-1)*(i-1)==n$
- B. $(i-1)*i==n$
- C. $i*i==n$
- D. $i*(i-1)==n$

答案: C

解析: 检查 n 是否为平方数, 条件是 $i*i == n$ (如 36 中 $i=6$) 。

④处应填 ()

- A. $n-i$
- B. $n-i+1$
- C. $i-1$
- D. i

答案: D

解析: 若 n 是平方数, 输出平方根 i (如 36 输出 6) 。

⑤处应填 ()

- A. $n / \text{fac}[k]$
- B. $\text{fac}[k]$
- C. $\text{fac}[k]-1$
- D. $n / (\text{fac}[k]-1)$

答案： A

解析： 逆序输出大于 \sqrt{n} 的因数，即 $n/\text{fac}[k]$ (如 12 的因数 3 对应 4) 。

完整版程序

```
#include <bits/stdc++.h>

using namespace std;

int main() {

    int n;

    cin >> n;

    vector<int> fac;

    fac.reserve((int)ceil(sqrt(n)));

    int i;

    for (i = 1; i * i < n; ++i) {

        if (n % i == 0) { //

            fac.push_back(i);

        }

    }

    for (int k = 0; k < fac.size(); ++k) {

        cout << fac[k]<< " "; //

    }

    if (i*i==n) { //

        cout << i<< " "; //

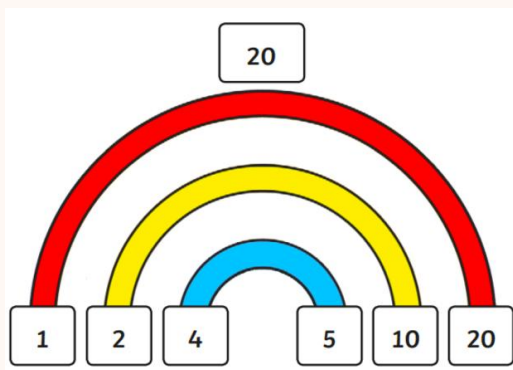
    }

    for (int k = fac.size() - 1; k >= 0; --k) {

        cout << n / fac[k]<< " "; //

    } }
```

36
1 2 3 4 6 9 12 18 36



1	×	36	=	36
2	×	18	=	36
3	×	12	=	36
4	×	9	=	36
6	×	6	=	36