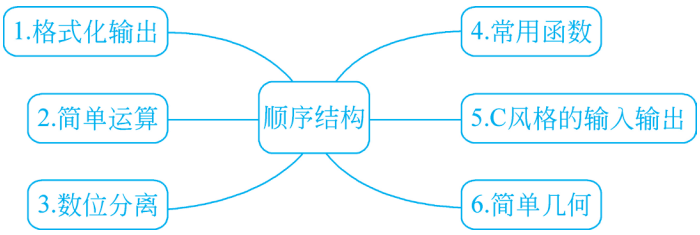
第3章　顺序结构

顺序结构是编程中最基本和最重要的控制结构。它指的是按照代码的书写顺序逐行执行代码的方式，顺序结构是编写程序的基础，几乎所有程序都包含顺序结构。

在顺序结构中，代码按照书写的顺序依次执行，不会跳过任何一行代码。这种结构通常用于执行一系列操作，例如计算数学公式、处理输入数据、输出结果等。在大多数编程语言中，程序从主函数（main 函数）开始执行，然后依次执行程序中的每条语句，直到执行完所有语句为止。

熟练掌握顺序结构对于编写有效且高质量的程序至关重要。它有助于我们清晰地组织代码，使程序易于阅读和维护，从而提高代码的可读性和可维护性。在前面章节的学习中，同学们已经基本掌握了一些编程技能。从本章开始，将以示例为引导，通过深入浅出的分析，并在关键步骤中提供相应的注释，帮助同学们更好地理解与掌握编程。



第 1 课　格式化输出

【导学牌】

理解输出格式在评测中的重要性

理解空格符、换行符在评测中的重要作用

【知识板报】

代码的评测依赖于输出数据的字符匹配,必须与测试点的要求严格一致。特别需要关注是否需要匹配相应的空格符或换行符。虽然大部分评测会忽略每行末尾的空格符和换行符,但在中间部分,必须严格根据要求输出符合规定格式的数据,这有助于培养细致和严谨的思维。

例 3-1:输出第二个整数

给定三个整数,请输出中间的第二个整数。

【输入格式】

三个整数 a、b 和 c,中间有空格间隔

【输出格式】

输出其中第二个整数

【输入样例】

```
3 5 2
```

【输出样例】

```
5
```

【数据范围】

$-10^9 \leqslant a, b, c \leqslant 10^9$

【示例代码】

解法 1:

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

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    cout << b << endl;
    return 0;
}
```

解法 2:

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

int main() {
    int a;
    cin >> a >> a;
    cout << a;
    return 0;
}
```

有两种常见的方法可以解决这个问题。解法 1 比较常规，需要创建三个变量，分别用来存储三个整数，然后输出第二个变量的值。

解法 2 相对来说更简洁，只需要定义一个变量，通过两次输入整数就可以直接进行输出。在算法竞赛中，方法多种多样，只要输出结果与测试数据一致即可。第二种方法在这里是一种巧妙的偷懒方式，它连续两次输入数据到变量 a，第二次输入数据的过程会覆盖掉原有 a 中的数据，使 a 最终保留了第二次输入的值，然后直接输出 a 即可。但是需要注意的是，如果还希望使用第一次输入的数据，那么就无法访问了。

这两种方法的输出结果相同，但第二种方法没有添加换行符。目前大多数评测机制会自动忽略行末的空格符和换行符，因此通常不会有问题。但若在一些编程网站中遇到年代久远的题目时，可能会因为行末没有添加换行符而导致无法通过。

例 3-2：芯芯拆盲盒

芯芯收到了 5 个盲盒，每个盲盒中都包含一个幸运数字。她按照从上到下的顺序依次打开了这些盲盒。然后，她将前 3 个盲盒中的幸运数字放在第一行，按照从左到右的次序排列。接着，她将剩下的两个盲盒中的幸运数字分别放在第二行和第三行。现在我们根据签收单，已经知道了从下往上的每个盲盒中的幸运数字。

请根据芯芯摆放幸运数字的策略，输出芯芯摆放的这三行幸运数字吧。

【输入格式】

5 个 int 类型的整数，以空格间隔

【输出格式】

三行幸运数，第一行 3 个幸运数之间有一个空格，后两行分别为 1 个幸运数字

【输入样例】

```
1 2 3 4 5
```

【输出样例】

```
5 4 3
2
1
```

【示例代码】

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

int main() {
    int a, b, c, d, e;
    cin >> a >> b >> c >> d >> e;
    cout << e << ' ' << d << ' ' << c << endl;
    cout << b << endl;
    cout << a << endl;
    return 0;
}
```

根据题意，示例代码的任务是根据芯芯打开盲盒的顺序，将输入的 5 个幸运数字根据对应的格式要求，输出成三行。

首先，使用 cin 语句依次读取签收单上的 5 个 int 型的整数，从下往上的幸运数字分别存储在变量 a、b、c、d、e 中。然后，使用 cout 语句按照从上往下的摆放策略输出这些幸运数字。根据策略，前 3 个幸运数字要按照从左到右排列在第一行，后两个幸运数字分别放在第二行和

第三行。

程序首先输出第一行的幸运数字,即 e、d、c,用空格隔开,然后换行。接着,程序输出第二行的幸运数字,即 b,然后换行。最后,程序输出第三行的幸运数字,即 a,然后换行。

程序运行后的数据输出,不仅数据要准确,格式也需完全匹配,才能够通过测评,例如第一行每个整数之间需要有空格,行与行之间需要通过 endl 实现换行等。

提示:输出空格的方法可以通过单引号或者双引号的方式,两者皆可。

例 3-3:芯芯的口算本

芯芯今天想练习口算,可是口算本丢在学校了,于是想到了使用计算机出题。现已知计算机会输入两个整数,以及算数符号,运算仅限于加法与减法,请输出这道完整的口算题。

【输入格式】

共有三行,前两行为两个整数,第三行为运算符号

【输出格式】

一道口算题的表达式,包括运算符和等于号,数字与符号之间有空格

【输入样例】

```
15
40
+
```

【输出样例】

```
15 + 40 =
```

【示例代码】

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

int main() {
    int a, b;
    char c;
    cin >> a >> b >> c;
    cout << a << ' ' << c << ' ' << b << " =" << endl;
    return 0;
}
```

cin 语句输入数据时会自动忽略空格和换行,即使间隔了很多的空格或者换行。这里不仅输入了两个整数,还输入了一个字符,并且每个变量之间要求有空格,最后在输出空格和等号的过程中,使用了双引号,将空格与等号作为一个整体,以字符串的形式输出。

由于篇幅的关系,本书无法提供每道题严格的输入格式、输出格式、输入样例、输出样例、数据范围、时间限制、内存限制等提示,尽管这些提示是每道题目中必备的一些标准参数,但在不影响理解题意的情况下,会略有精简,详细的题目信息可以通过在线测评 OJ 平台查阅。

第2课　简单运算

【导学牌】

理解数据范围与超限

掌握基于输入数据的简单运算

【知识板报】

编程的运算与数学运算联系紧密，虽然有一些区别，例如编程中符号的样式与数学运算中的略有不同，但基本的运算功能保持一致。

变量的存储会占用一定的内存空间，每个数据类型有它表示的范围，并不是无限大的，在运算过程中，要注意数据是否存在超限的可能。

例 3-4：A + B Problem

给定两个整数 a 和 b，请计算它们的和。

【输入格式】

一行，两个整数

【输出格式】

一行，一个整数

【输入样例】

```
120 400
```

【输出样例】

```
520
```

【数据范围】

$-10^9 \leqslant a, b \leqslant 10^9$

【示例代码】

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

int main() {
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
    return 0;
}
```

这是一道经典的入门题，输入两个变量后进行加法运算，在此过程中值得注意的是数据范围，两个变量的范围是－10 亿～10 亿之间，可以是负数。在程序的评测中，一定要考虑最极端的情况，这里两个数据都是 10 亿，相加后也是在 int 型范围内，此题定义两个 int 型的变量是满足题目要求的，但是如果数据范围保持不变，实现三个数的相加，由于 int 型数据表示的最大值大约是 20 亿多一点，30 亿显然会超过范围，则需要使用 long long 型变量，具体的数据范围在前文有所介绍，可以时常查阅。

例 3-5：A * B Problem

给定两个整数 a 和 b，请计算它们的乘积（$-50{,}000 \leqslant a, b \leqslant 50{,}000$）。

【输入样例】

```
3 9
```

【输出样例】

```
27
```

【示例代码】

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

int main() {
    long long a, b;
    cin >> a >> b;
    cout << a * b << endl;
    return 0;
}
```

在最值情况下,按题目中出示的数据范围,计算结果会超限,例如 50000 * 50000,结果已经超过了 int 型数据的表示范围,对于部分数据范围较大的测试点,程序评测中会判定为错误,因此可以使用 long long 型变量,以确保计算结果在数据范围内。

例 3-6:带余除法

给定两个正整数 a 和 b,请计算它们的商与余数,并用空格隔开($0<b\leqslant a\leqslant 2\times 10^9$)。

【输入样例】

```
7 3
```

【输出样例】

```
2 1
```

【示例代码】

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

int main() {
    int a, b;
    cin >> a >> b;
    cout << a / b << ' ' << a % b << endl;
    return 0;
}
```

根据数据提示,因为两个数据都是正整数,所以不用考虑 b 为 0 的特殊情况。在 C++编程中,整除运算使用运算符“/”,模运算使用取模运算符“%”,从而可以轻松地求出两个整数的商与余数,注意中间有空格间隔。

第3课　数位分离

【导学牌】

学会数位分离的基本方法

理解前导零的概念

【知识板报】

数位分离是整数处理中的基本操作，它通过整除和求余两种运算相结合的方式实现。其目标是分离整数各位上的数字，以便进行单独的处理或分析，数位分离的操作常常用于解决涉及数字操作与统计的相关问题。

前导零是指位于一个数字的最高位左侧的零，这些零并不改变数字的值，但会影响数字的表示形式。通常，前导零的添加是为了对齐位数，或者满足特定格式的要求。

例 3-7：整数的反转

给定一个四位整数，请反向输出各位数，并计算各位数之和。

【输入样例】

```
2458
```

【输出样例】

```
8 5 4 2
19
```

【示例代码】

解法 1：

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

int main() {
    int x, a, b, c, d;
    cin>>x;
    d = x % 10;
    c = x / 10 % 10;
    b = x / 100 % 10;
    a = x / 1000;
    cout<<d<<' '<<c<<' '<<b<<' '<<a<<endl;
    cout<<a + b + c + d;
    return 0;
}
```

解法 2：

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

int main() {
    char a, b, c, d;
    cin>>a>>b>>c>>d;
    cout<<d<<c<<b<<a<<endl;
    cout<<(a - '0') + (b - '0') + (c - '0') + (d - '0')<<endl;
    return 0;
}
```

解法 1,采用了数位分离的经典思路,是从右往左进行逐位分离。为了获取某一位上的数字,可以通过除法移位将该位的数值移到个位上,然后通过模运算(% 10)得到该位的数值。

解法 2,根据字符输入的特性,通过建立 4 个字符型变量并逐个输入字符,实现了反向输出。在求和过程中,通过将字符减去字符'0',将其转换为对应的整数,从而进行求和操作。这种方法利用了字符的 ASCII 码与对应的数字字符之间的关系,使得字符能够直接转换为整数进行运算。

两种方法没有优劣之分,只是使用了不同的策略,可以根据遇到的不同情景,选择相对更适合的解法。

例 3-8:三位数的首尾互换

给定一个可能含有前导零的三位数 X($0 \leqslant X \leqslant 10^9$),请将它的首尾两位数进行交换并输出,交换后的数字需隐藏前导零。

【输入样例 1】

```
012
```

【输出样例 1】

```
210
```

【输入样例 2】

```
140
```

【输出样例 2】

```
41
```

【示例代码】

解法 1:

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

int main() {
    int x, a, b, c;
    cin >> x;
    c = x % 10; x /= 10;
    b = x % 10; x /= 10;
    a = x % 10;                    // % 10 运算也可以不写,因为 x 已经是个位数
    cout << c * 100 + b * 10 + a;
    return 0;
}
```

解法 2:

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

int main() {
    int x, y = 0;
    cin >> x;
    y = y * 10 + x % 10;
    x /= 10;
    y = y * 10 + x % 10;
    x /= 10;
    y = y * 10 + x % 10;
```

```
        x /= 10;
        cout << y << endl;
        return 0;
    }
```

解法 1 的思路和例 3-7 解法 1 基本一致，仍然是从右往左进行数位分离。在分离的过程中，每次通过模 10 的运算操作保留个位的数值，并通过 x 除以 10 的方式去掉当前的个位数字。由于需要避免前导零，分离后的数据需要按位加权求和，以得到最终的反向数字。在主函数中，有的行出现了两句话，虽然不会影响编程的运行，但通常不建议在一行中出现以分号结尾的两句话。在这里这样编写是为了突出变量 x 的实际变化和规律。

解法 2 相对于解法 1 更为简洁，只需要两个变量即可完成任务。首先，声明了两个整数变量 x 与 y，x 用于存储输入的整数，y 用于存储反向后的整数，初始值为 0。接下来，通过一系列的操作，逐位地从 x 中提取数字，并将它们添加到 y 的末尾，然后从 x 中去除个位数字。这一过程将被执行三次，每次都将 x 的个位数字添加到 y 的末尾，然后将 x 右移一位以去除个位数字。最终，当 x 变为 0 时，y 中存储的即为输入整数的反向数字。

第4课　常用函数

【导学牌】

理解相关函数的特性与适用场景

掌握常用函数的运用方法

【知识板报】

函数并不是一种高深复杂的概念,它们的存在旨在简化和方便程序的设计与使用。初学者无需对函数感到畏惧,可以将它们看作一种具有某种功能的工具。应该欢迎并灵活运用一些常见的函数,因为这可以大大提高编程效率。

本课介绍的函数包含在<cmath>头文件中。这个头文件是C++标准库中的数学库头文件,提供了各种数学函数和常量,可用于数学运算。其中包括绝对值函数、取整函数、幂函数、三角函数、指数函数、对数函数等。

由于万能头文件<bits/stdc++.h>已经包含了<cmath>头文件,在编写代码时无须单独添加<cmath>头文件,在此仅作为介绍。

例3-9:绝对值

芯芯在玩一个游戏,游戏一共有3轮,每轮从抽奖盒里面抽取两个球,每个球上有一个数字X(0≤X≤100),每轮游戏的积分是两个球上的数字差值,请计算3轮游戏后,芯芯的总积分是多少。

【输入样例】

```
1 3
30 20
7 99
```

【输出样例】

```
104
```

【示例代码】

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

int main() {
    int a, b, sum = 0;
    cin >> a >> b;
    sum += abs(a - b);
    cin >> a >> b;
    sum += abs(a - b);
    cin >> a >> b;
    sum += abs(a - b);
    cout << sum << endl;
    return 0;
}
```

首先将变量sum初始化为0,接着通过3次数据的输入,进行累加后求得最后的总积分。在这个过程中,涉及数学函数abs(),它用于求绝对值。在abs()函数的括号中,如果数据是负数,会将其转换为正数,否则将保持原值不变。此外,还有另一个与之类似的fabs()函数,用于

求浮点数的绝对值，需要注意的是，这两者之间存在一些区别。

abs()函数通常用于求整数的绝对值，而 fabs()函数用于求浮点数的绝对值。尽管在一些C++标准中，abs()函数也支持对浮点数求绝对值，但为了代码的可读性和明确性，推荐根据使用场景的不同，选择使用 abs()或 fabs()函数。

例 3-10：取整

给定一个浮点数 X($-10^6 \leqslant n \leqslant 10^6$)，请分别对它进行向上取整、向下取整、四舍五入这三种运算，并逐行输出。

【输入样例】

```
2.65
```

【输出样例】

```
3
2
3
```

【示例代码】

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

int main() {
    double x;
    cin >> x;
    cout << ceil(x) << endl;
    cout << floor(x) << endl;
    cout << round(x) << endl;
    return 0;
}
```

这里使用了三个常用的函数，分别是向上取整 ceil()函数，向下取整 floor()函数，四舍五入 round()函数。

ceil()函数，用于向上取整，返回不小于给定参数的最小整数值(向正无穷方向取整，类型是 double)。例如 ceil(3.5)的结果为 4，ceil(−0.5)的结果为−0，编程中是存在负零这种表现形式的，但注意 ceil(1.0)的结果依然为 1。

floor()函数，用于向下取整，返回不大于给定参数的最大整数值(向负无穷方向取整，类型是 double)。例如 floor(1.1)的结果为 1，floor(−2.1)的结果为−3。如果对一个非负浮点数取整，默认就是向下取整，例如 int(2.9)的结果为 2，不过要注意 floor()函数返回的结果是浮点型。

round()函数，用于标准的四舍五入，它将浮点数四舍五入到最接近的整数(类型是 double)。在实战中，非负浮点数的四舍五入有更简单的方式，可以手动将原始数据加 0.5，再将结果自动取整，取整后会自动去掉小数点，可以在不涉及特定舍入规则的情况下，快速而简单地实现四舍五入的操作，例如 int(1.5+0.5)的结果为 2，int(2.45+0.5)的结果为 2。

除了标准的四舍五入，还有其他的一些舍入规则，例如银行家舍入，遵从四舍六入五成双的规则，round()函数在有些编程语言中，比如 python 等，默认遵从银行家舍入规则，有兴趣的同学可以自行研究。

例 3-11：幂运算

给定两个整数 a 和 b($1 \leqslant a, b \leqslant 9$)，请计算 a 的 b 次方，并输出整数型表示的结果。

【输入样例 1】

```
2 10
```

【输出样例 1】

```
1024
```

【输入样例 2】

```
9 8
```

【输出样例 2】

```
43046721
```

【示例代码】

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

int main() {
    int a, b;
    cin >> a >> b;
    cout << int(pow(a, b));
    return 0;
}
```

pow()函数用于计算一个数的指数幂,该函数接受两个参数,分别为底数和指数,函数返回的结果为浮点型。如果数位在 6 位有效数字内,会直接显示,如果超过 6 位有效数字,会采用科学记数法,例如 pow(9, 8)默认显示为 4.30467e+07。由于结果需要用整数型表示,所以要将浮点型结果转换成整数型再输出。

对于幂运算的结果,尤其是多次幂运算或指数较大的运算情况下,由于浮点数的精度限制,返回值可能会引发误差,导致最终结果不准确。因此在特定的应用需求和精度要求下,需要选择适当的计算方法,确保最大程度地减小误差的影响。

例 3-12:物归原主

芯芯和同学们正在玩一个数字交换的游戏。她们一共有 5 个人,站成一个圆圈。游戏规则如下:从第一个人开始,每个人都要将手上的数字先与左边的朋友交换,然后再与右边的朋友交换,如此往复,直到每个人都完成了交换,最终所有人的数字,会物归原主。

现给定 5 个人的原始数字 A_i($1 \leqslant A_i \leqslant 1000$),请输出每个人在完成交换后,5 个人的数字情况。

【输入样例】

```
1 2 3 4 5
```

【输出样例】

```
2 5 3 4 1
5 3 2 4 1
5 2 4 3 1
5 2 3 1 4
1 2 3 4 5
```

【示例代码】

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

```
int main() {
    int a1, a2, a3, a4, a5;
    cin >> a1 >> a2 >> a3 >> a4 >> a5;
    swap(a1, a5); // a1 与左边的 a5 交换数值
    swap(a1, a2); // a1 与右边的 a2 交换数值
    cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << ' ' << a5 << endl;
    swap(a2, a1);
    swap(a2, a3);
    cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << ' ' << a5 << endl;
    swap(a3, a2);
    swap(a3, a4);
    cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << ' ' << a5 << endl;
    swap(a4, a3);
    swap(a4, a5);
    cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << ' ' << a5 << endl;
    swap(a5, a4);
    swap(a5, a1);
    cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << ' ' << a5 << endl;
    return 0;
}
```

在示例代码中，使用了 swap()交换函数，该函数接受两个参数，其功能是交换这两个参数变量的数值。相对于手动实现两个变量的数值交换，使用 swap()函数会更加方便，而且不容易出错。

swap()函数不仅适用于整数型变量的交换，而且适用于几乎所有基本数据类型，包括整数型、浮点型、字符型、自定义数据类型等。交换的前提是这两个变量必须属于相同的数据类型，只有相同数据类型的两个变量才能够被成功交换。如果两个变量的数据类型不一致，例如 int 型和 long long 型这两种类型的整数变量，那么它们无法直接使用 swap()函数进行交换，需要采取额外的措施，如手动转换类型或使用中间变量来实现。

例 3-13：最大值与最小值

给定一行用空格间隔的 3 个整数 a，b，c($1 \leqslant a, b, c \leqslant 10^9$)，请比较它们的大小关系，并根据以下要求进行输出，共两行。

第一行输出两个数据，用空格间隔：1. 输出前两个整数的较大值，2. 输出 3 个整数的最大值。

第二行输出两个数据，用空格间隔：1. 输出前两个整数的较小值，2. 输出 3 个整数的最小值。

【输入样例】

```
7 9 5
```

【输出样例】

```
9 9
7 5
```

【示例代码】

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

int main() {
```

```
    int a, b, c;
    cin >> a >> b >> c;
    cout << max(a, b) << ' ' << max(max(a, b), c) << endl;
    cout << min(a, b) << ' ' << min(min(a, b), c) << endl;
    return 0;
}
```

max()与 min()函数的功能分别是查找两个数值中的最大值与最小值,包含在< algorithm >头文件中。函数可以接受两个参数,返回其中的较大值或较小值。如果需要比较两个以上的数据,则可以通过函数的嵌套使用实现,例如求 4 个数据的最大值,可以使用多种比较的方法,如先找到前两位的最大值,再和后两位的最大值进行对比,最终返回 4 个数据中的最大值,当然还可以有其他的嵌套比较方法,示例如下。

```
int maxn = max(max(a, b), max(c, d));
int maxn = max(max(max(a, b), c), d);
```

在 C++11 及以后的标准中,max()和 min()函数还有另外一种形式,如 max({a, b, c}),在函数中如果引入了大括号,可以比较更多的元素值,有兴趣的同学可以自行研究。

第 5 课　C 风格的输入输出

【导学牌】

理解 C 风格在程序设计中的优势与特点

掌握 C 风格输入输出的基本格式

【知识板报】

C 风格，是指 C 编程语言的代码风格和语法规则。C++ 继承了 C 语言的大部分特性和语法，C 风格的代码在 C++ 中仍然有效。

在之前的学习中，我们已经掌握了通过 C++ 的 cin 和 cout 语句实现标准输入和输出操作。现在，让我们来探讨一下 C 风格的输入和输出，主要通过 scanf() 函数和 printf() 函数来实现。学习 C 风格的输入输出有其必要性，因为相对于 cin 和 cout 语句，C 风格在输入输出大量数据时具有更高的效率。在处理大量数据时，为了避免程序运行超时，C 风格通常更为适用。

例如，当需要读取空格或者处理相对复杂的数据格式时，scanf() 函数会更加方便。同样，当需要按照特定格式进行输出时，printf() 函数也具有明显的优势，比如控制小数点后的位数、设置输出宽度、补充前导零等，这些都可以更轻松地实现。

虽然 C 风格有很多优点，但对于初学者来说，可能需要记忆一些格式化符号与变量地址的操作。这些细节可以在日后的学习中逐渐掌握，不必急于一时。

这里梳理了常用的 C 风格占位符，其中 n 代表数字，具体如表 3-1 所示。

表 3-1　常见的 C 风格占位符

占位符	输入	输出	功　能
%d	√	√	整数的输入与输出
%nd	√	√	输入或输出 n 位整数，输出位数不足时用空格补齐
%0nd	√	√	输入或输出 n 位整数，输出位数不足时用数字 0 补齐
%c	√	√	字符的输入与输出
%lf	√	√	双精度浮点数的输入与输出
%f	√	√	单精度浮点数的输入与输出
%lld	√	√	long long 类型整数的输入与输出
%.nf		√	保留小数点后 n 位的输出
%s	√	√	字符数组的输入与输出

例 3-14：买菜时间

芯芯的奶奶每天都喜欢逛菜场。芯芯记录了今天奶奶的出门时间，以及奶奶预计买菜所需的时间(以分钟为单位)。芯芯需要计算出奶奶大概何时能够回家。

要注意的是，时钟以 24 小时制表示。出门时间和购物时间都以时和分表示，如 06:25 表示早上 6 点 25 分。购物所需时间以分钟表示，如 55 表示 55 分钟。

目前的任务是需要计算奶奶的回家时间，输出的结果需要包括时和分，并确保时和分都占据两个位置，如果需要，用数字 0 来补齐。结果保证在同一天内。

请根据输入的出门时间和购物所需时间，输出奶奶预计回家的时间。

【输入样例】

```
06: 25
55
```

【输出样例】

```
07: 20
```

【示例代码】

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

int main() {
    int h, m, delta, time;
    scanf("%d:%d %d", &h, &m, &delta);
    time = h * 60 + m + delta;
    printf("%02d:%02d", time / 60, time % 60);
    return 0;
}
```

根据题目要求,需要计算奶奶预计回家的时间。示例代码使用了 scanf()函数进行输入,按照"时:分"的格式读取了奶奶出门的时间以及购物所需的时间 delta。

在输入部分,程序通过 scanf()函数按照特定的格式读取了输入数据。双引号中的格式字符串包含了占位符%d,每个占位符对应后面部分的一个变量。在示例代码中,3 个%d 占位符分别对应 h、m、delta 3 个变量。前两个%d 之间有一个冒号,用于匹配输入数据中的冒号,而后两个%d 之间的空格则用于匹配读入过程中的空格。需要注意的是,在 C 风格的数据读入中,变量前需要加上符号 &,以确保正确读入数据,否则会读入数据失败。

在计算部分,首先将小时转换为分钟,通过乘 60 的方式实现,然后加上分钟和购物所需的时间 delta,这一步骤非常清晰,确保了奶奶回家时间的准确性。

最后使用 printf()函数输出结果,其中%02d 格式的占位符确保输出的小时和分钟都占据两个位置,并在需要时用数字 0 补齐,这种输出格式非常友好,使结果更容易阅读。

例 3-15:芯芯的测试成绩

芯芯参加了学校的期中和期末两次测试,每次测试都包括语文、数学和英语三门科目。现已知她在这两次测试中的成绩(均为整数)。学校将综合评分作为评选三好学生的主要依据,综合评分由期中和期末测试的两次成绩综合计算而来。其中,期中成绩占总评成绩的 40%,期末成绩占总评成绩的 60%。

请分别计算以下内容:

1. 期中测试的总分与平均分(保留两位小数)
2. 期末测试的总分与平均分(保留两位小数)
3. 综合评分(保留两位小数)

【输入样例】

```
95 98 99
97 100 98
```

【输出样例】

```
292 97.33
295 98.33
293.80
```

【示例代码】

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

int main() {
    int ch1, ch2, ma1, ma2, en1, en2, sum1, sum2;
    scanf("%d %d %d\n%d %d %d", &ch1, &ma1, &en1, &ch2, &ma2, &en2);
    sum1 = ch1 + ma1 + en1;
    sum2 = ch2 + ma2 + en2;
    printf("%d %.2f\n", sum1, 1.0 * sum1 / 3);
    printf("%d %.2f\n", sum2, 1.0 * sum2 / 3);
    printf("%.2f", sum1 * 0.4 + sum2 * 0.6);
    return 0;
}
```

示例代码通过 C 风格的输入一次性读取了两行内容，分别记录了期中和期末测试的成绩。然后通过计算，统计了每次测试的总分并进行输出。在这个过程中，如果将总分直接除以 3 会导致结果是整数，而我们需要保留两位小数。为了确保输出的是浮点数，在整数之前使用了 1.0 相乘的方式。这样，总分在计算过程中会自动转换为浮点数，从而使得除法运算的结果也是浮点数，最终通过占位符%.2f 得到带有两位小数的输出。

在程序中出现了转义字符“\n”，表示换行，在 C++中还有一些其他的转义字符，如“\r” “\t”等。这些字符在文本处理中具有特殊的作用，有助于控制输入和输出的格式，感兴趣的同学可以自行研究。

除了采用 C 风格的方式提高读入与输出的效率，还可以在 C++的主函数中通过关闭同步功能提高程序的效率，具体方法如下，感兴趣的同学可以自行研究。

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);                    // 或 cin.tie(nullptr);
    cout.tie(0);                   // 或 cout.tie(nullptr);

    return 0;
}
```

第6课　简单几何

【导学牌】

了解常见的几何概念

掌握几何问题中常见的计算方法

【知识板报】

信息学中经常会出现与几何相关的问题,例如计算几何、图形处理、空间关系判断等。掌握常见的几何概念、计算方法和算法技巧等,可以帮助解决这些几何问题,提高解题的准确性和效率。

解决几何问题需要综合运用数学、逻辑推理、编程和算法等多个领域的知识和技能,利用编程实现跨领域的实践,能够提升解决问题的能力与综合思维。

例 3-16:矩形的周长与面积

给定矩形的两个邻边 a 与 b(1≤a,b≤10000),请编写程序计算并输出该矩形的周长与面积。保证数据输入的边长均为正整数。

【输入样例】

```
5 95
```

【输出样例】

```
200 475
```

【示例代码】

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

int main() {
    int a, b;
    cin >> a >> b;
    cout << 2 * (a + b) << ' ' << a * b;
    return 0;
}
```

根据周长和面积的计算方法,可以模拟计算过程以得出相应的结果。在进行计算时,需要注意数据的范围,特别是在极值情况下。例如,如果两个边长都为 10000,结果仍然在 int 型范围内,不会产生问题。然而,如果范围进一步扩大,应考虑是否需要使用 long long 型来保存结果,以确保在更大的数值范围内计算结果的准确性。

例 3-17:圆的直径、周长与面积

给定一个圆的半径 r(1≤r≤5000),编写程序计算并输出该圆的直径、周长和面积。保证输入的半径 r 为正整数。

输出要求:直径输出为整数,周长和面积的输出保留 4 位小数,圆周率统一为 3.1415926。

【输入样例】

```
10
```

【输出样例】

```
20
```

```
62.8319
314.1593
```

【示例代码】

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

int main() {
    int r, d;
    double c, s;
    const double PI = 3.1415926;
    cin >> r;
    d = 2 * r;
    c = 2 * PI * r;
    s = PI * r * r;
    printf("%d\n%.4f\n%.4f", d, c, s);
    return 0;
}
```

在数学中，通常使用符号 r 表示圆的半径，d 表示直径，c 表示周长，s 表示面积。在示例代码中，通过使用 C 风格的输出方法来保留 4 位小数，这对于确保输出结果的准确性和格式一致性非常方便。

为了避免意外的输出错误，极为重要的一点是保持占位符与变量类型的一致性。例如，使用占位符%d 输出整数，使用占位符%.4f 输出保留 4 位小数的浮点数，如果前后不一致会出现意想不到的输出而导致结果错误。

例 3-18：几何距离

给定平面上两个点的坐标(x1, y1)和(x2, y2)，编写程序计算并分行输出这两个点之间的欧几里得距离、曼哈顿距离和切比雪夫距离，三者均保留两位小数。

【输入样例】

```
3 4
8 9
```

【输出样例】

```
7.07
10.00
5.00
```

【示例代码】

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

int main() {
    int x1, x2, y1, y2;
    double dis1, dis2, dis3;
    cin >> x1 >> y1 >> x2 >> y2;
    dis1 = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
    dis2 = abs(x1 - x2) + abs(y1 - y2);
    dis3 = max(abs(x1 - x2), abs(y1 - y2));
    printf("%.2f\n%.2f\n%.2f\n", dis1, dis2, dis3);
    return 0;
}
```

我们需要掌握以下三种距离的定义：

1. 欧几里得距离：又称直线距离，是最常见的几何距离。

2. 曼哈顿距离：又称城市街区距离，是在平面上两个点之间沿网格线(垂直和水平方向)行进的最短距离，是两点的横坐标和纵坐标之间的差值之和。

3. 切比雪夫距离：又称棋盘距离，是两点横坐标和纵坐标之间的最大差值。

在示例代码中，sqrt()函数是一个非常有用的工具，它能够计算给定数值的平方根。在计算欧几里得距离时，通常需要计算两个点之间的距离，这涉及计算每个坐标轴上的差值的平方，然后将它们相加，最后再取平方根，以得出最终的距离。其他两类距离，可以根据定义和有关数据进行相应的计算。

这三种距离度量方法在计算机科学和数据分析领域经常被使用，根据具体的问题和应用场景，选择合适的距离度量方法可以提供有效的数据分析和决策支持。

例 3-19：海伦公式

给定一个三角形的三条边(浮点数)，分别为 a、b 和 c(1≤a，b，c≤1000)，求出该三角形的面积，并保留两位小数输出。

【输入样例】

```
3
5.1
7.91
```

【输出样例】

```
3.33
```

【示例代码】

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

int main() {
    double a, b, c, p, s;                              // p 为半周长
    cin >> a >> b >> c;
    p = (a + b + c) / 2;                               // 计算半周长
    s = sqrt(p * (p - a) * (p - b) * (p - c));  // 海伦公式
    printf("%.2f", s);
    return 0;
}
```

示例代码演示了如何使用海伦公式计算三角形的面积。海伦公式是一个用于计算三角形面积的数学公式，它可以在已知三角形每条边长的情况下，直接计算出三角形的面积，而无需知道三角形的高或顶点的角度等信息。用编程的书写方式，公式为：s＝sqrt(p＊(p－a)＊(p－b)＊(p－c))，其中半周长 p＝(a＋b＋c)/2。

程序通过输入获取了三角形的三条边长，分别存储在变量 a、b 和 c 中。接着计算三角形的半周长 p，这是海伦公式中的一个关键参数，半周长的计算方式是将三条边的长度相加，然后除以 2。最终通过计算，使用 printf()函数以保留两位小数的格式输出计算得到的三角形面积。

这段示例代码对于计算任何已知三边长度的三角形的面积都是通用的，因此非常实用。此示例代码展示了如何在 C++中使用数学公式来解决实际问题，特别是与几何相关的问题。

本章寄语

在本章中，我们学习了如何进行数据的标准输入和输出，理解了只有符合标准的格式化输出才能通过评测。在此基础上，我们学习了一些简单的数学运算以及需要注意的细节，通过将数字分解成各个位数的形式提交，我们更深入地理解了评测机制。在编程竞赛中，虽然采取的方法不同，但只要得到的输出数据能与测试点匹配，都能够获得满分，这是评测机制决定的。

本章还介绍了一些常见的函数，为我们后续的学习奠定了良好的基础。掌握常见函数可以帮助我们快速实现相关的应用和功能。在第 2 章，我们学习了通过 cin 语句来实现数据的输入，在本章中，我们学习了 C 风格的输入与输出，通过示例演示，充分了解了其使用方法及优势。C 风格中有关输入与输出的占位符需要识记，以便在日后的实践中能够灵活运用。

我们运用到了几何学和数学的结合，如计算周长和面积，计算不同种类的距离等，希望能够帮助同学们将已掌握的各学科知识应用到新的领域，这种迁移和应用知识的能力对编程竞赛非常重要。