第二章 数据类型与顺序结构程序设计

本章内容

- 2.1 变量的数据类型
- 2.2 变量的输入与输出
- 2.3 顺序结构程序设计案例
- 2.4 提交评测与错误自查

2.1 变量的数据类型

核心知识

数据类型决定变量能够保存什么数据、占用多少存储空间，以及参与运算时采用什么规则。基础课程中常用的数据类型包括整数、浮点数、字符和布尔值。

分类	类型	主要用途	示例
整数	`short`	较小范围整数(-128~127)	`short level = 10;`
整数	`int`	中范围整数，**最常用**(-21亿~21亿)	`int age = 15;`
整数	`long long`	大范围整数(非常大，万兆以上)	`long long total = 80000000;`
浮点数	`float`	单精度小数	`float rate = 0.5;`
浮点数	`double`	双精度小数，**小数最常用**	`double price = 12.5;`
字符	`char`	单个字符	`char grade = 'A';`
布尔值	`bool`	真或假	`bool passed = true;`

选择类型时要同时考虑数据范围和计算精度。题目可能出现小数时，一般使用 `double`；整数范围可能很大时，需要考虑 `long long`。

`char`、转义字符与 ASCII

`char` 用来保存单个字符，字符值必须使用单引号：

```
char letter = 'A';
char digit = '7'; // 数字字符，不是整数 7
char op = '*';
char blank = ' ';
```

字符串使用双引号。`'A'` 是一个字符，`"A"` 是一个字符串，两者不能混用。

字符在内存中以整数编码保存。基础题通常使用 ASCII 规律处理英文字母和数字字符：

字符范围	ASCII 值
`'0'` ~ `'9'`	48 ~ 57

字符范围	ASCII 值
`'A'` ~ `'Z'`	65 ~ 90
`'a'` ~ `'z'`	97 ~ 122

常用转换：

```
char c = '8';
int number = c - '0';              // 数字字符转整数，得到 8

char upper = 'B';
char lower = upper + ('a' - 'A');  // 大写字母转小写字母，得到 'b'

int code = 65;
char letter = (char)code;           // 强制转换后得到字符 'A'

char x = 'A';
int ascii = (int)x;                 // 强制转换后得到整数 65
```

转义序列以反斜杠 `\` 开头：

写法	含义
`\n`	换行
`\t`	水平制表符(Tab键, 一般代表的 4 个空格)
`\\`	一个反斜杠
`\"`	一个双引号
`\'`	一个单引号

示例：

```
cout << "小明说:\"what is \\n\"" << '\n';
```

输出：

```
小明说:"what is \n"
```

类型转换

当表达式中出现不同数据类型时，编译器可能进行自动类型转换。一般而言，整数与浮点数混合运算时，整数会转换为浮点类型：

```
int a = 5;
double b = 2.0;
double result = a / b; // 2.5
```

把 `double` 转为 `int` 会丢弃小数部分：

```cpp
double value = 3.9;
int n = (int)value; // 3，不进行四舍五入
```

如果要转换整个表达式，应把表达式完整括起来：

```cpp
char result = (char)(a + b);
```

典型例题

题目： 输入一个大写英文字母，输出对应的小写字母。

```cpp
int main() {
    char upper;
    cin >> upper;

    char lower = (upper - 'A') + 'a'; // 也可通过加减 32 来实现，大写转小写需要加 32
    cout << lower;
    return 0;
}
```

说明： ASCII 中大写字母和小写字母各自连续排列，而对应大小写字母的差值则一定相同。

常见错误

- 把单个字符写成 `"A"`。`char` 值应使用单引号 `'A'`。
- 混淆整数 `7` 与字符 `'7'`。`'7' - '0'` 才得到整数 7。
- 让 `int` 保存需要保留小数的计算结果。
- 强制类型转换前没有考虑数据范围和精度损失。
- 直接假设任何字符都能通过加减 32 转换大小写。该规律只适用于对应的英文字母。
- 想输出一个反斜杠时只写一个反斜杠。字符串中应写 `"\\"`。

2.2 变量的输入与输出

核心知识

`cin`、`cout` 是 C++ 的流式输入输出工具，通过 `<iostream>` 使用。`scanf`、`printf` 是格式化输入输出函数，需要添加 `<cstdio>` 使用。

输入多个数据时，变量的数量、类型和顺序必须与题意一致。

输出时不仅要保证答案数值正确，还要严格匹配题目要求的空格、换行、小数位数和文字内容。

`cin` 与 `cout`

```cpp
int a, b;
cin >> a >> b;           // 依次读入两个整数
cout << a << " " << b;   // 输出两个整数，中间一个空格
cout << a + b << '\n';   // 输出计算结果并换行
```

使用规则：

- `cin` 使用提取运算符 `>>`，把输入保存到变量。
- `cout` 使用插入运算符 `<<`，把数据发送到控制台。
- `cin >> a >> b;` 会跳过前导的空白字符，输入中的空格、Tab、回车通常都能分隔数据。
- `cout << "a+b";` 输出文本 `a+b`； `cout << a + b;` 输出计算结果。
- `endl` 和 `'\n'` 都能换行。

`scanf` 与 `printf`

```
#include <cstdio>

int age;
double height;
char grade;

scanf("%d%lf %c", &age, &height, &grade);
printf("%d %.2f %c\n", age, height, grade);
```

常用格式说明符：

数据类型	`scanf` 读取	`printf` 输出
`int`	`%d`	`%d`
`long long`	`%lld`	`%lld`
`float`	`%f`	`%f`
`double`	`%lf`	`%f`
`char`	`%c`	`%c`

输出小数位数：

```
double value = 3.1415926;
printf("%.2f\n", value); // 3.14
printf("%.3f\n", value); // 3.142
```

`scanf` 要把数据写入变量，因此变量名前通常要加取地址符 `&`：

```
int n;
scanf("%d", &n);
```

格式字符串中的普通字符必须与输入对应：

```
scanf("%d%d", &a, &b);   // 可输入：10 20
scanf("%d,%d", &a, &b);  // 必须输入：10,20
```

`%c` 默认不会跳过空白字符。需要跳过已有的空格或换行时，可在 `%c` 前留一个空格：

```
char op;
scanf(" %c", &op);
```

典型例题

题目： 输入总水量 `t`（毫升）和人数 `n`。输出每人分到的水量，保留 3 位小数；第二行输出需要的杯子数，每人 2 个杯子。

```
int main() {
    double t;
    int n;
    scanf("%lf%d", &t, &n);

    printf("%.3f\n", t / n);
    printf("%d\n", n * 2);
    return 0;
}
```

说明： `scanf` 读取 `double` 使用 `%lf`；`printf` 输出 `double` 使用 `%f`；`%.3f` 表示保留 3 位小数。

常见错误

- 把方向写反：`cin` 使用 `>>`，`cout` 使用 `<<`。
- `scanf` 漏写 `&`，导致无法把输入正确保存到变量。
- 格式说明符与变量类型不匹配。此问题不只是“精度降低”，还可能导致读取失败或未定义行为。
- 混淆 `double` 的格式：`scanf` 读取用 `%lf`，`printf` 输出用 `%f`。
- `scanf("%d,%d", ...)` 要求输入中真的出现逗号。
- 用 `%c` 读取字符时，意外读入了前一次输入留下的空格或换行。
- OJ 输出中多了提示语、单位、标点或空格。

2.3 顺序结构程序设计案例

核心知识

顺序结构程序按照代码书写顺序逐条执行。基础题通常可以分成“输入、处理、输出”三层：

```
// 1. 输入
cin >> ...;

// 2. 处理
// 计算公式、更新变量、转换单位

// 3. 输出
cout << ...;
```

面对程序设计题时，可以使用七步法：

1. **读懂题目**：明确最终要计算什么。
2. **理解输入**：确认数据的数量、类型、顺序和单位。

3. **设计逻辑思路**：先写公式或处理步骤。
4. **翻译成代码**：定义变量，把思路逐步写成 C++ 语句。
5. **处理输出**：安排顺序、空格、换行和精度。
6. **编译查错**：先解决语法错误。
7. **样例验证**：输入样例并对照输出，再补充边界数据。

常见解题方法包括：

- 把复杂计算拆成多个中间结果，用变量保存。
- 统一不同单位，再进行计算。
- 使用临时变量，防止原数据在赋值时被覆盖。
- 在关键位置临时输出中间变量，与手算结果逐步核对。

交换变量时需要第三个变量中转：

```
int a = 3, b = 4;
int temp = a;
a = b;
b = temp;
```

例题：小鱼的游泳时间

题目： 小鱼从当天 `a` 时 `b` 分游到 `c` 时 `d` 分。输入四个整数，输出游泳时长的小时数和分钟数。本题保证结束时间不早于开始时间。

```
#include <iostream>
using namespace std;

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

    int startMinutes = a * 60 + b;
    int endMinutes = c * 60 + d;
    int duration = endMinutes - startMinutes;

    int hours = duration / 60;
    int minutes = duration % 60;

    cout << hours << " " << minutes;
    return 0;
}
```

思路说明：

1. 输入开始时间和结束时间，共 4 个整数。
2. 把两个时刻统一换算为分钟。
3. 用结束总分钟数减去开始总分钟数。
4. 用 `/ 60` 得到小时，用 `% 60` 得到剩余分钟。
5. 输入 `12 50 19 10` 时，应输出 `6 20`。

如果结果错误，可以临时输出中间变量：

```
cout << "start=" << startMinutes << '\n';
cout << "end=" << endMinutes << '\n';
cout << "duration=" << duration << '\n';
```

提交 OJ 前要删除这些调试输出。

补充案例：按格式反转数字

输入形如 `123.4` 的固定格式数据，将其输出为 `4.321`。可以把输入看作 4 个字符和一个固定的小数点：

```
#include <cstdio>

int main() {
    char a, b, c, d;
    scanf("%c%c%c.%c", &a, &b, &c, &d);
    printf("%c.%c%c%c\n", d, c, b, a);
    return 0;
}
```

这里的 `.` 是格式字符串中的普通字符，因此输入中也必须在对应位置出现小数点。

常见错误

- 把样例数据直接写死，程序无法处理其他合法输入。
- 时间差分别计算小时差和分钟差，却没有处理借位。
- 交换变量时直接写 `a = b; b = a;`，使原值丢失。
- 复杂公式全部写在一行，出错后难以定位。
- 只测试样例，不测试整点、临界值、最小值等情况。
- 调试输出未删除，导致答案格式不匹配。

2.4 提交评测与错误自查

提交流程

1. 在本地完成代码并成功编译。
2. 输入题目样例，确认输出内容和格式一致。
3. 把完整源代码提交到 OJ。
4. 根据评测结果定位问题，修改后重新测试和提交。

OJ 只检查程序输出的字符序列是否符合要求。不要擅自添加输入提示、答案说明或单位。

常见评测结果

结果	含义	优先检查
AC（Accepted）	通过全部测试数据	程序已通过本题评测
CE（Compile Error）	编译错误	分号、括号、变量名、头文件、中文标点

结果	含义	优先检查
WA（Wrong Answer）	答案错误	公式、条件、类型、边界情况、输出格式
RE（Runtime Error）	运行时错误	除数为 0、非法访问、错误输入函数用法
TLE（Time Limit Exceeded）	超出时间限制	算法效率、循环终止条件

错误自查顺序

出现 CE

1. 查看编译器给出的第一条错误信息和行号。
2. 检查该行以及前一行是否漏写分号。
3. 检查 `()`、`{}`、引号是否成对。
4. 检查变量是否先定义后使用，名称是否拼写一致。
5. 检查是否误用了中文标点或缺少头文件。

出现 WA

1. 先检查输出格式：空格、换行、小数位数、文字大小写。
2. 用草稿纸手算样例和一组自选数据。
3. 分步输出中间变量，找出第一个与手算不一致的位置。
4. 检查整数除法、数据类型和运算优先级。
5. 检查最小值、最大值、临界值和特殊情况。

出现 RE(目前较少遇见)

1. 检查除数是否可能为 0。
2. 检查 `scanf` 的格式说明符和 `&`。

本章练习

一、基础知识

1. 为下列数据选择合适的类型：学生人数、商品价格、考试等级字符、条件真假、较大的整数总数。

2. 说明字符 `'7'` 与整数 `7` 的区别。

3. 写出 `'0'`、`'A'`、`'a'` 的 ASCII 值。

4. 写出以下转换表达式：
   - 数字字符 `ch` 转为对应的整数数字。
   - 大写英文字母 `upper` 转为对应的小写字母。

- 整数 `code` 强制转换为字符。

5. 写出下列转义序列的作用：`\n`、`\t`、`\\`、`\"`。

6. `scanf` 读取 `int`、`double`、`char` 时分别使用什么格式说明符？

7. `printf` 输出 `double` 并保留两位小数时，格式说明符应怎样写？

8. `scanf` 中为什么通常要在变量名前加 `&`？

9. 分别说明 AC、CE、WA 的含义。

二、代码阅读与改错

1. 写出程序的输出结果：

```
char c = '6';
int n = c - '0';
cout << n + 4;
```

2. 写出程序的输出结果：

```
char upper = 'C';
char lower = upper + ('a' - 'A');
cout << lower;
```

3. 找出格式说明符的错误并改正：

```
int age;
double height;
scanf("%lf%d", &age, &height);
printf("%d %.2d\n", age, height);
```

4. 找出变量交换失败的原因并改正：

```
int a = 3, b = 8;
a = b;
b = a;
cout << a << " " << b;
```

三、程序设计

1. **字符与 ASCII**

   输入一个字符，输出该字符以及它对应的 ASCII 值，中间用一个空格分隔。

   - 样例输入：`A`
   - 样例输出：`A 65`

2. **数字字符转整数**

   输入一个 `'0'` ~ `'9'` 的数字字符，将它转换成整数后加 5，并输出结果。

   - 样例输入：`7`
   - 样例输出：`12`

3. **格式化平均值**

   输入三个 `double` 数据，输出它们的平均值，结果需要严格保留 3 位小数。