03 运算符与表达式

求值规则及其结果

先认识表达式

先建立定义和学习目标，再比较具体表达式的结果

表达式

- 表达式是 C 程序中能够按规则计算出结果的代码片段。这个计算过程称为求值。
- 42 和 sample_count 本身都是表达式
- 3 + 4 用运算符把两个表达式组合成新的表达式
- 有些表达式在求值时还会改变变量的当前值
- 这里先用这个定义识别表达式。实例之后再补充类型、组合和状态变化的边界。

完成本单元后，你应当能够

- 拆分常见表达式，说明各步的类型和值
- 根据任务选择整数除法、浮点除法或余数运算
- 追踪表达式引起的状态变化，并用运行结果核对预测

从三个结果开始

先作出预测，再用本单元建立的模型逐项验证

先预测三个结果

先预测，不运行

```
1 7 / 4
2 7.0 / 4
3 (double)(7 / 4)
```

每个结果是什么类型?

哪一步最先丢失了小数部分?

本单元将按同一分析顺序回答

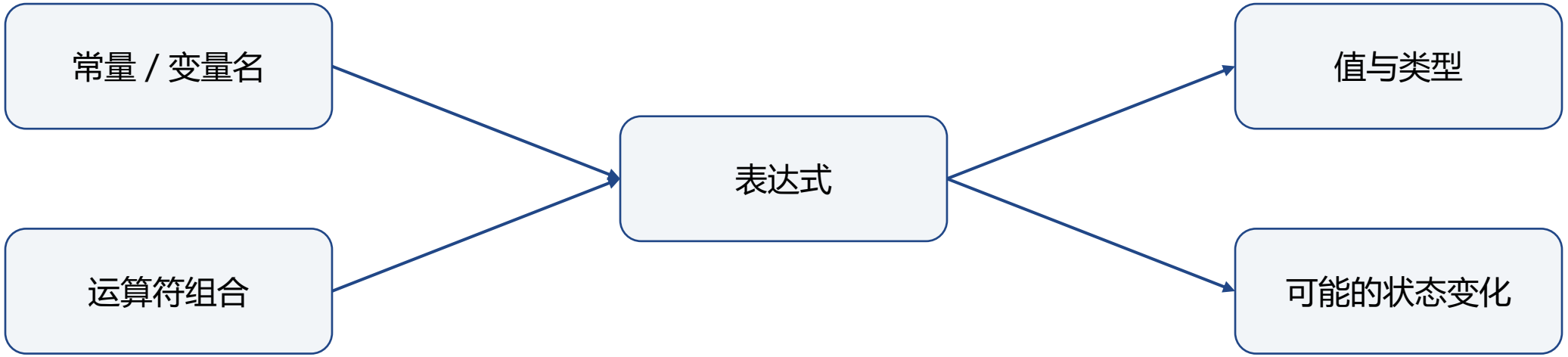
分析表达式时依次回答

1 表达式怎样产生值

建立共同模型，再解释组合计算

表达式的完整定义

- **表达式**是具有类型且可以按 C 规则求值的语言构造
- 常量和变量名本身可以构成表达式
- 运算符可以把已有表达式组合成新的表达式
- 本单元只讨论求值得到值的常见表达式
- 赋值和更新表达式在求值时还可能改变程序状态



先拆分，再预测

expression_trace.c

```c
#include <stdio.h>

int main(void)
{
    int a = 8;
    int b = 3;
    int result = a + b * 2;

    printf("%d\n", result);
    return 0;
}
```

第 1 步：_____

第 2 步：_____

哪些变量改变：_____

两步得到结果 14

expression_trace.c

```c
#include <stdio.h>

int main(void)
{
    int a = 8;
    int b = 3;
    int result = a + b * 2;

    printf("%d\n", result);
    return 0;
}
```

b * 2 → 3 * 2 → int 值 6

a + 6 → 8 + 6 → int 值 14

a、b 的当前值不变

括号、优先级与结合性

- **括号**
  - 明确指定怎样组合
  - 是直接的可读性工具
- **优先级**
  - 乘除余高于加减
  - 只决定表达式分组
- **结合性**
  - 同级连续出现时决定分组
  - 连续的加减或乘除按从左向右结合

分组不等于实际求值先后

- 优先级和结合性回答“表达式怎样组合”
- 它们不自动回答所有操作数按什么先后求值
- 状态变化可能相互影响时，拆成多条清楚语句

2 除法的结果由类型决定

操作数类型不同，运算规则可能不同

开场三个表达式的结果

开场问题

```
1 7 / 4
2 7.0 / 4
3 (double)(7 / 4)
```

int 值 1

double 值 1.75

double 值 1.0

商与余数共同描述整数分组

- 对非负整数 7 和 4：
- $7 = (7/4) \times 4 + (7\%4)$
- **商** 1：完整的 4 有多少个
- **余数** 3：完整分组后剩下多少
- % 的两个操作数必须是整数类型

3671 秒怎样拆成时、分、秒

time_conversion.c

```c
#include <stdio.h>

int main(void)
{
    int total_seconds = 3671;
    int hours = total_seconds / 3600;
    int remaining = total_seconds % 3600;
    int minutes = remaining / 60;
    int seconds = remaining % 60;

    printf("%d:%02d:%02d\n", hours, minutes, seconds);
    return 0;
}
```

当前规则的使用边界

- 商与余数示例只使用非负整数
- 除数不能为 0
- 需要小数结果时，参与除法的两个操作数中至少一个是浮点类型
- 用 0、59、60、3599、3600 检查分解结果

3 转换必须发生在关键运算以前

不仅问转换成什么，还要问何时转换

转换太晚，丢失已经发生

conversion_position.c

```c
#include <stdio.h>

int main(void)
{
    int total = 7;
    int count = 4;
    double late = (double)(total / count);
    double early = (double)total / count;

    printf("late  = %.2f\n", late);
    printf("early = %.2f\n", early);
    return 0;
}
```

late

整数除法 1
→ 再转为 1.0

early

先把 total 转为 double
→ 浮点除法 1.75

转换位置决定哪一次运算采用新类型

显式转换表达程序意图

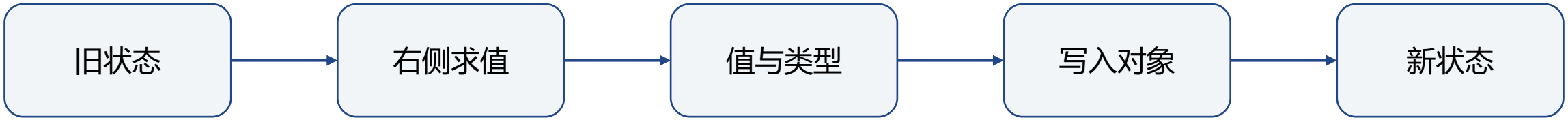
- 合理使用
  - 为当前运算选择需要的类型
  - 在需要改变规则的运算发生前完成转换
- 不能做到
  - 恢复此前丢失的小数部分
  - 证明单位、取值范围和任务含义正确

4 赋值把结果写回状态

有些表达式在产生值的同时改变变量当前值

赋值连接计算与状态

- 表达式求值时，除了产生值，还可能改变程序状态。这种状态变化称为**副作用**。



右侧先计算，左侧再更新

state_update.c

```c
#include <stdio.h>

int main(void)
{
    int score = 10;
    int bonus = 3;

    score = score + bonus * 2;
    printf("%d\n", score);

    score += bonus;
    printf("%d\n", score);
    return 0;
}
```

初始：score = 10，bonus = 3

第一次更新：score = 16

第二次更新：score = 19

bonus 始终保持 3

递增写法保持简单

- 推荐
  - count++;
  - ++count;
  - 独立成句时都把 count 增加 1
- 避免
  - 把多个状态变化塞进更大的表达式
  - 根据某次输出猜测未说明的求值先后

5 用预测与运行验证计算

先按提示追踪，再独立修正错误

分析表达式的五个问题

1. 写清分组
2. 标出操作数和子表达式的类型
3. 逐步预测值
4. 标出可能改变的状态
5. 编译并运行程序，比较预测与实际输出，再用边界输入检查

修正平均值计算

repair_average.c

```c
#include <stdio.h>

int main(void)
{
    int total = 17;
    int count = 5;
    double average = total / count;

    printf("%.2f\n", average);
    return 0;
}
```

当前结果 17 / 5 → int 3 → 3.00

先在纸上写出最小修改，再运行程序核对。

不要只报告 3.40；还要解释哪一步改变了运算类型。

在除法以前完成转换

repair_average.c

```c
#include <stdio.h>

int main(void)
{
    int total = 17;
    int count = 5;
    double average = total / count;

    printf("%.2f\n", average);
    return 0;
}
```

当前结果 17 / 5 → int 3 → 3.00

最小修改 (double)total / count

转换必须发生在整数除法以前

独立任务：拆分总秒数

- 输入一个非负总秒数，计算完整小时、剩余完整分钟和最后剩余秒数。
- 同时写出 0、59、60、3599、3600 的预期结果。

附录：本单元与教材的对应

- 《C Primer Plus》第 5 章
  - 基本运算符、表达式与优先级：必读
  - 除法、求模和类型转换：必读
  - 递增、递减和副作用：选读
  - 循环、函数和完整语句分类：后续单元
- 习题编号待采用版本确认

本单元总结：分析表达式的五个问题

本单元学习检查

学习内容	检查标准
表达式拆分	分组与 C 规则一致
值与类型预测	能解释整数 / 浮点结果差异
状态追踪	能区分不改变状态的计算与变量更新
修错结果	修改发生在最早错误位置
边界测试	预期与输出一致，并能解释差异