01 初识 C 程序

从源代码到运行结果

1 让代码产生结果

从熟悉的面积任务出发，先预测结果，再观察代码产生的实际结果

本讲学习目标

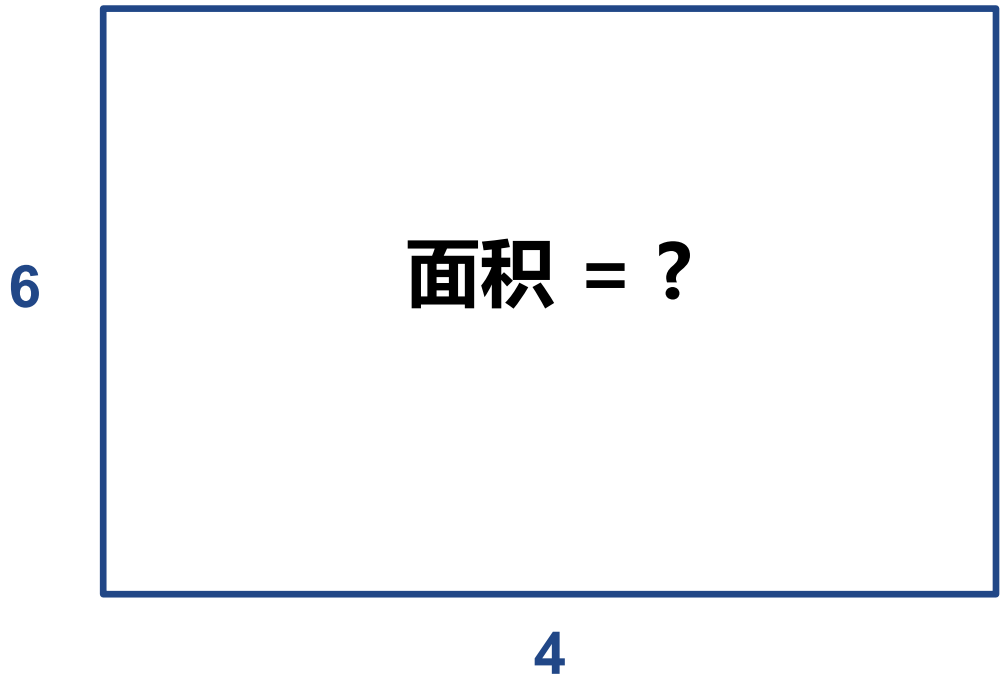
- 根据任务预测结果，运行代码并比较实际结果
- 根据代码的执行位置，说明长、宽和面积数据的变化
- 代码无法运行或结果不符合任务要求时，根据提示或实际结果修改代码，然后再次验证
- 本讲先理解完整过程，不要求记忆全部 C 语言语法规则。

第一个任务：计算面积

任务

已知矩形的长是 4，宽是 6

目标是计算并显示面积 24



先确定预期，再观察程序

直接计算 4 × 6

```c
#include <stdio.h>

int main(void)
{
    printf("%d\n", 4 * 6);
    return 0;
}
```

观察目标

先找到要计算的算式

完整代码

放在左侧，先保持整体可见

`4 * 6`

当前只观察这个算式

第一次运行

```c
#include <stdio.h>

int main(void)
{
    printf("%d\n", 4 * 6);
    return 0;
}
```

运行结果

屏幕显示 24

计算依据

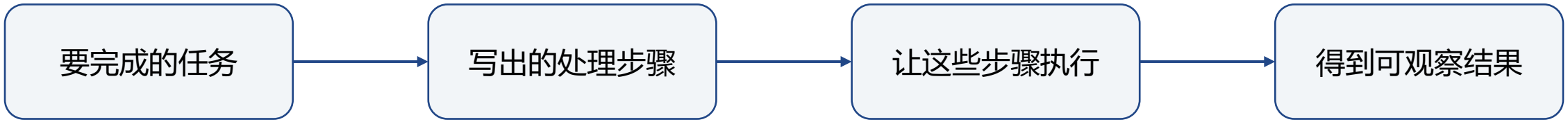
4 × 6

printf

把结果显示出来

任务、代码和结果

- 任务要求计算并显示 4 × 6
- 代码写出了计算和显示的步骤
- 执行这些步骤后，屏幕显示 24



程序与源代码

- **程序**是一组按一定顺序组织、让计算机完成任务的明确指令
- 程序员把处理步骤写成 C 语言文本，这份文本称为 **源代码**
- 运行程序时，计算机按照已经构建好的程序处理本次数据并产生结果

2 读懂并扩展这份程序

认识完整程序的基本组成，再让同一程序处理不同数据

先读懂三个位置

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%d\n", 4 * 6);
6     return 0;
7 }
```

main

当前程序从这里开始执行

{ ... }

围出 main 中依次执行的语句

printf(...)

计算并显示本例的结果

其余写法先识别

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%d\n", 4 * 6);
6     return 0;
7 }
```

#include <stdio.h>

使编译器能够识别本例使用的 printf

%d 与 \n

显示整数并在之后换行

return 0;

表示当前程序正常结束

固定数据的局限

4 × 6

修改代码中的数字

↓

重新构建

3 × 5

再次修改代码

↓

再次构建

每换一组数据，都要改代码并重新构建

同一份代码处理不同长宽

```c
#include <stdio.h>

int main(void)
{
    int width = 0;
    int height = 0;
    int area = 0;

    printf("Enter width and height: ");
    scanf("%d%d", &width, &height);
    area = width * height;
    printf("Area = %d\n", area);

    return 0;
}
```

数据路径

输入 → 保存 → 计算 → 显示

输入

scanf 把两个数写入 width 和 height

保存

三个变量保存本次运行中的数据

计算

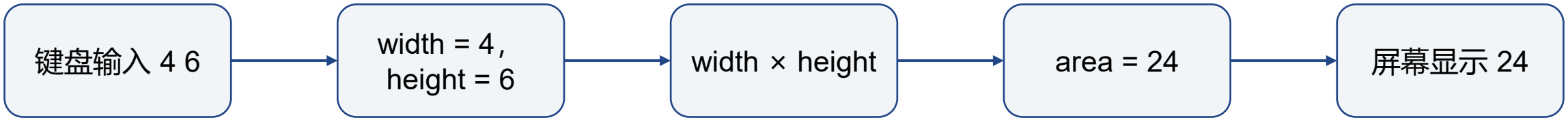
area = width * height

显示

printf 显示 area 的当前值

输入与变量

- 本次运行交给程序处理的数据称为 **输入**
- **变量**是有名称、能够在运行时保存当前数据的对象
- 输入 4 和 6 后，width 和 height 保存这两个数；area 随后保存计算结果 24



3 观察程序怎样执行

从静态阅读转向状态预测，再用调试器逐步核对

某一时刻的程序状态

- 程序执行到不同位置时，变量的当前值可能不同
- 本讲用执行位置、相关变量的当前值和已经产生的结果描述 **程序状态**
- 这是本讲采用的简化描述，不等于程序在计算机中的全部状态

先预测三个时刻的状态

先填写问号处的预测，再用调试器逐行核对

程序位置	width	height	area
输入之前	0	0	0
输入完成之后	?	?	?
面积计算之后	?	?	?

用调试器观察程序

调试器是暂停执行、逐步运行并观察当前状态的工具

先观察正确程序，再用它核对自己的预测

输入前的变量

```c
#include <stdio.h>

int main(void)
{
    int width = 0;
    int height = 0;
    int area = 0;

    printf("Enter width and height: ");
    scanf("%d%d", &width, &height);
    area = width * height;
    printf("Area = %d\n", area);

    return 0;
}
```

观察时刻

输入之前

width = 0

尚未取得输入

height = 0

尚未取得输入

area = 0

面积语句尚未执行

输入后的变量

```c
#include <stdio.h>

int main(void)
{
    int width = 0;
    int height = 0;
    int area = 0;

    printf("Enter width and height: ");
    scanf("%d%d", &width, &height);
    area = width * height;
    printf("Area = %d\n", area);

    return 0;
}
```

观察时刻

输入完成之后

`width = 4`

保存输入得到的长

`height = 6`

保存输入得到的宽

`area = 0`

面积语句尚未执行

计算后的 area

```c
#include <stdio.h>

int main(void)
{
    int width = 0;
    int height = 0;
    int area = 0;

    printf("Enter width and height: ");
    scanf("%d%d", &width, &height);
    area = width * height;
    printf("Area = %d\n", area);

    return 0;
}
```

观察时刻

面积计算之后

`width = 4`

保存输入得到的长

`height = 6`

保存输入得到的宽

`area = 24`

由 width * height 计算得到

4 从源代码到可执行程序

已经看见运行时状态，现在区分源代码怎样被构建和运行

输入变化与代码变化

改变输入

输入 3 和 5

↓

直接再次运行

↓

得到 15

改变源代码

把乘法改为加法

↓

重新构建

↓

运行新程序

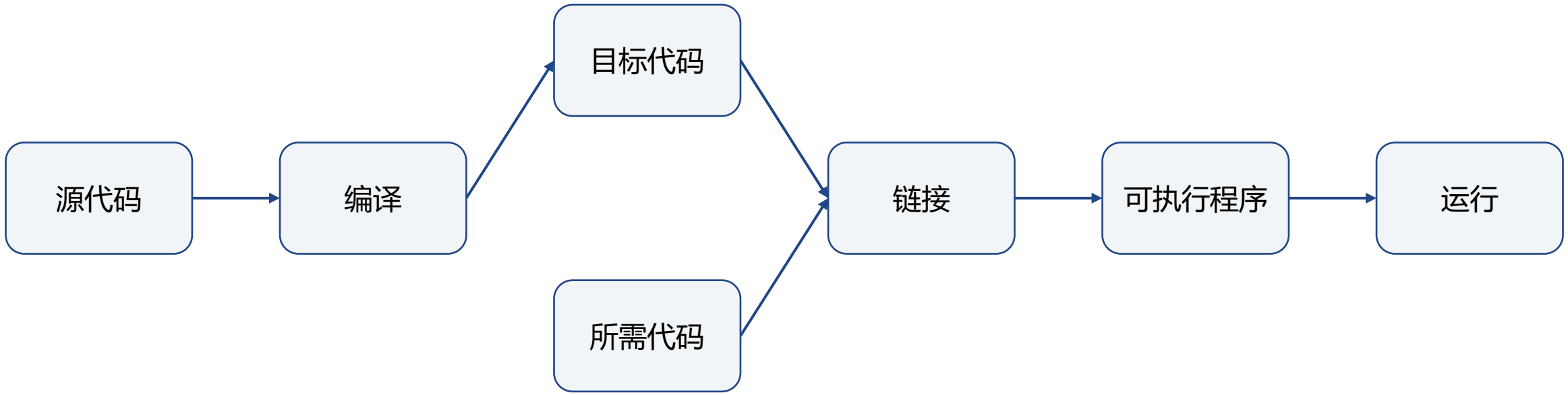
输入改变本次数据，源代码改变程序的处理方式

构建：让新代码成为可运行程序

- **构建**是把源代码转变为对应可执行程序的过程
- 修改源代码以后，需要重新构建，才能得到与新代码对应的 **可执行程序**
- 课程环境中的 Build 会连续完成本讲需要认识的编译和链接

构建中的编译与链接

- **编译**检查并翻译当前源文件，生成可供链接处理的结果
- 这个编译结果也称为 **目标代码**
- **链接**把目标代码与程序所需的其他代码连接成可执行程序



5 发现并修正问题

正常程序已经能够构建和运行，现在依据不同证据定位问题

缺少分号：无法构建

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int width = 0;
6      int height = 0;
7      int area = 0;
8
9      printf("Enter width and height: ");
10     scanf("%d%d", &width, &height);
11     area = width * height
12     printf("Area = %d\n", area);
13
14     return 0;
15 }
```

构建结果

编译没有完成

诊断线索

编译器报告语句不完整

检查位置

从诊断位置附近开始

原因

这里缺少分号

先判断：这段程序会发生什么

```c
#include <stdio.h>

int main(void)
{
    int width = 0;
    int height = 0;
    int area = 0;

    printf("Enter width and height: ");
    scanf("%d%d", &width, &height);
    area = width + height;
    printf("Area = %d\n", area);

    return 0;
}
```

先判断

暂不运行，也不查看下一页

构建

这段代码能否通过构建?

结果

输入 4 和 6 后会显示什么?

任务

显示结果是否符合矩形面积要求?

编译成功也可能算错

```c
#include <stdio.h>

int main(void)
{
    int width = 0;
    int height = 0;
    int area = 0;

    printf("Enter width and height: ");
    scanf("%d%d", &width, &height);
    area = width + height;
    printf("Area = %d\n", area);

    return 0;
}
```

判断

实际结果与预期不一致

预期

4 × 6 = 24

实际

4 + 6 = 10

编译结果

代码形式正确，能够完成编译

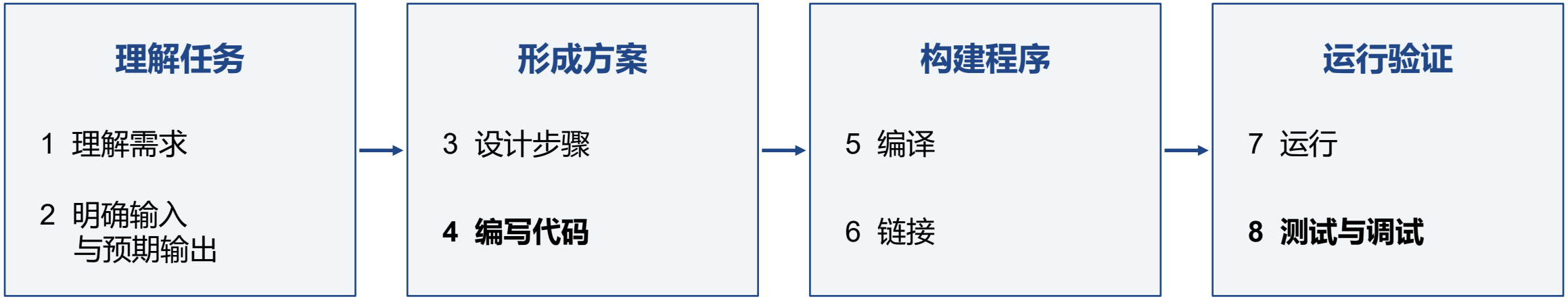
任务判断

结果不符合矩形面积要求

从差异开始调试

- 先明确程序应该得到什么结果
- 再用运行、诊断和变量状态收集证据
- 每次修改后都要重新验证结果

编程的基本过程



编程是反复理解、实现和检验的过程

本讲总结

- 程序员用源代码表达处理步骤，通过构建得到可执行程序。运行可执行程序时，程序根据本次输入产生结果
- 在当前顺序程序中，输入改变变量的当前值。执行位置、相关变量的当前值和已经产生的结果共同描述本讲范围内的程序状态
- 编译诊断可以提供构建问题的线索。程序能够运行时，仍需根据任务预期检查结果
- 遇到差异时，先明确预期，再观察证据、定位修改并重新验证