C++ 顺序结构程序综合实战与解题方法 讲义

1. 引言

- **问题引入：我们学会了什么？我们要做什么？**
  我们已经学会了如何定义变量、运算符和输入输出。已经可以写出一些基本的顺序结构程序了，我们现在要用这些工具，去综合利用解决一个实际问题。而很多同学在面对一个编程题时，会感觉大脑一片空白，不知道第一行代码该怎么写。这很正常！因为目前还缺少一套系统的解题流程。
- **价值阐述：为什么要学习解题流程？**
  一个优秀的程序员，强大的不仅仅是他的代码能力，更是清晰的**逻辑思维和解决问题的能力**。今天要学习的这套流程，就是一个思维框架。它能帮助你：
  1. **克服做题的恐惧**：面对任何题目，你都有一个可以遵循的“套路”，不会再手足无措。
  2. **提高效率**：将一个大问题，分解成多个可控的小步骤，避免在错误的方向上浪费时间。
  3. **提升正确率**：系统性地思考，减少逻辑漏洞。

---

2. 顺序程序结构综合练习：如何交换两个变量的值

- **问题：如何交换 num1 与 num2 变量里的值**:

```
int num1 = 3, num2 = 4;
// ... 在这里写代码 ...
cout << num1 << " " << num2 << endl; // 目标：输出 4 3
```

- **错误的尝试**: 很多同学的第一反应是这样的：

```
num2 = num1; // num2 的值变成了 3
num1 = num2; // num1 的值变成了 num2 (也就是3) 的值
```

  问题出在哪里？ 当你执行 `num2 = num1;` 时，`num2` 原本的值 `4` 就被 `num1` 的值 `3` **覆盖**掉了，永远地丢失了！就像你把一瓶可乐倒进了一瓶装有雪碧的瓶子里，雪碧就没了。

- **正确的解法：引入第三个“空杯子”**
  为了不丢失任何一个值，我们需要一个临时的变量（我们称之为 `temp` 或 `num3`）来做中转。

  例：num1为可乐，num2为雪碧，尝试进行交换：
  1. `int temp = num1;` (把 num1 的可乐，倒进空杯子 temp 里)
  2. `num1 = num2;` (把 num2 的雪碧，倒进 num1 杯里, num1 变成了雪碧)
  3. `num2 = temp;` (把 temp 杯里的可乐，倒进 num2 杯里, num2 变成了可乐)
  4. **至此，交换成功！**

- **代码实现**:

```
int num1 = 3, num2 = 4;
int temp = num1; // 引入第三个变量
num1 = num2;   num2 = temp; // 使用空杯子中转一下
cout << num1 << " " << num2; // 输出 4 3
```

---

3. 编程解题的“七步法”

现在，让我们以“**小鱼的游泳时间**”这个问题为案例，一步一步地走完我们解题的完整流程。

第一步：读懂题目，明确题目要求“做什么”？

- **原文**：小鱼从 a 时 b 分一直游泳到当天的 c 时 d 分，计算它一共游了多少时间。
- **提炼任务**：核心任务是**计算时间差**。我们要用“结束时间”减去“开始时间”。

第二步：理解输入，明确需要输入的原材料是什么？

- **原文**：输入四个整数 a, b, c, d。
- **解读**：程序需要从外界接收4个整数。我们必须定义4个 `int` 类型的变量来“接住”这些数据。

```
int a, b, c, d;
cin >> a >> b >> c >> d;
```

- **重要误区**：绝对不能看到样例输入是 `12 50 19 10` 就在代码里写 `int a = 12;`！样例只是一个例子，我们的程序必须能处理任何合法的输入。**变量的值必须通过 `cin` 来获取！**

第三步：思考解题逻辑，先想出解决问题的思路

在敲代码之前，先把思路理出来。清晰的逻辑思路，远比一开始就埋头敲错误的代码要高效得多。

- **实际的问题**：如何计算 `c时d分 - a时b分`？直接减很麻烦，比如 `19时10分 - 12时50分`，分钟不够减，需要向小时借位。
- **寻找更简单的方法 -> 统一单位**：
  所有复杂的多单位计算，都可以通过**转换成统一的最小单位**来简化。这里的最小单位是“分钟”。
  1. **将所有时间转换为分钟**：
     - 开始时间（总分钟数）`start = a * 60 + b;`
     - 结束时间（总分钟数）`end = c * 60 + d;`
  2. **计算差值**：
     - 现在游泳的总时长就是开始时间和结束时间的差值 `diff = end - start;`。
  3. **将结果转换回目标格式**：
     - 题目要求输出“e小时 f分钟”。我们手上有总分钟数，如何转换回去？
     - 小时数 `e = diff_minutes / 60;` (利用整数除法自动取整)
     - 分钟数 `f = diff_minutes % 60;` (利用取余运算得到不够1小时的剩余部分)

第四步：把思路翻译为C++代码”

现在，我们把第三步的逻辑，一行行地用C++语言表达出来。

```
// 步骤1：统一单位
int start = a * 60 + b;
int end = c * 60 + d;
int diff = ends - start;// 步骤2：计算差值

// 步骤3：转换格式
int e = diff / 60; // 小时
int f = diff % 60; // 分钟
```

第五步：处理输出，保证格式绝对正确

- **原文**：一行内输出两个整数 e 和 f，用空格间隔。
- **解读**：在线评测系统(OJ)要求输出必须和标准答案一模一样，一个空格都不能差。
- **代码实现**:

```
cout << e << " " << f;
```

第六步：编译运行，检查语法错误

把所有代码组合起来，在你的编程环境中（如Dev-C++）按下编译运行键（F11）。

- 如果出现 `Compile Error (CE)`，仔细阅读编译器给出的错误信息，它会告诉你哪一行出错了，通常是漏了分号、括号不匹配、变量名写错等失误。

第七步：样例测试，验证逻辑的正确性

- **操作**：程序运行后，输入题目给的样例 `12 50 19 10`，然后按回车。
- **比对**:
  - 程序输出：`6 20` 并且题目的样例输出：`6 20`
  - **结果完全一致！** 这说明我们的逻辑在处理这个样例时是正确的。
- **如果结果不一致怎么办？ -> 使用手动 cout 检查法**
  假设你的程序输出了奇怪的结果，比如 `1 -30`。这说明逻辑错了！
  1. **手动在关键位置加入`cout`**：在你代码的关键步骤后，通过`cout`打印出关键的中间变量的值。

```
int start = a * 60 + b;
cout << "开始总分钟: " << start << endl; // 实际输出出来看看 start 的值到底是多少，跟我想的是一致的吗
int end = c * 60 + d;
cout << "结束总分钟: " << end << endl;   // 实际输出出来看看 end 的值到底是多少，跟我想的是一致的吗
int diff = end - start;
cout << "分钟差: " << diff << endl;      // 实际输出出来看看最终的结果差的底是多少，跟我自己算出来的是一致的吗
```

  2. **重新运行样例查看输出结果**：输入 `12 50 19 10`。
  3. **进行核对**:
     - 通过实际在草稿纸上计算：如果输入样例数据`12 50 19 10`的话，那`start`就应该是`12*60+50=770`，`end`应该是`19*60+10=1150`，`diff`应该是`380`。
     - 查看你的程序输出，如果某个`cout`打印的值和实际算出来不一致，你就能知道问题出现在什么地方了！**就要仔细进行检查。**

---

4. OJ的反馈结果

在我们的题目网站上递交你的代码后，最终它会给你一个结果来告诉我们你的代码是否正确。我们就要学会读懂OJ的反馈的结果。而系统的反馈主要有这几种：

1. 正确`AC (Accepted)`

- **含义**：**正确通过！**恭喜！意味着程序已经通过了所有测试数据。

2. 编译错误：`CE (Compile Error)`

- **含义**：**编译错误。**
- **解读**：这是最基础的错误。它意味着你的代码存在**语法问题**，翻译官根本无法读懂你的代码。
- **常见的编译错误原因**：
  1. **漏写分号 (`;`)**：每一句指令的结束标志，就像中文的句号。
  2. **括号不匹配**：`{}` 和 `()` 总是成双成对出现，少了一半就会出错。
  3. **使用了中文标点**：代码世界里只认英文标点。中文的 `，；（）` 都是非法字符。
  4. **变量名拼错**：你定义时用的是 `score`，使用时却写成了 `scroe`。
  5. **变量未定义就使用**：还没准备好盒子（声明变量），就想往里面放东西。
- **如何解决**：
  `CE` 是最容易解决的问题。**在DevC++编程软件中仔细查看错误信息**，它通常会告诉你**哪一行**出错了，并给出一个英文的错误原因。根据提示回到你的代码中修改，直到在本地能够成功编译为止。

3. 逻辑错误：`WA (Wrong Answer)`

- **含义**：**答案错误。**`WA` 意味着你的**程序语法没问题**，也能顺利运行，但在某一组测试数据上，你的程序给出的输出结果和标准答案不一致。这说明，你的**程序设计本身存在逻辑漏洞。**
- **如何自查和更正？**：
  1. **第一步：检查输出格式**
     - 你的 `cout` 语句里是不是多了不该有的东西？比如 `cout << "答案是：" << sum;`。
     - 你的输出里是不是少了题目要求的空格或换行？比如要求 `6 20`，你输出了 `620`。
  2. **第二步：检查你的解题思路**
     - 如果格式没问题，那一定是逻辑思路出了错。
     - 拿出草稿纸，重新走一遍你的“七步做题法”中的第三步。你的思路真的能应对所有情况吗？是不是对题目的某个细节理解有偏差？
  3. **第三步：使用 `cout` 手动检查法**
     - 在代码的关键计算步骤之后，插入临时的 `cout` 语句，打印出中间变量的值进行检查。
     - 用比如题目给出的样例来运行你的的程序。观察打印出来的中间值，和你在草稿纸上一步步推算的结果进行比对。一旦发现哪个变量的值与你的预期不符，就找到了程序“不正确的位置！就可以仔细进行检查并修正。

---

6. 总结

请务必牢记着七个编程的基本步骤：

读懂题目(识别核心任务是什么) → **理解输入** (明确输入数据的类型、数量、顺序及格式，并据此在程序中声明相应变量) → **思考逻辑** (设计解决问题的思路) → **翻译代码** (将逻辑思路翻译为具体的C++代码) → **处理输出** (将答案根据题目要求的格式输出) → **编译查错** (检查是否有语法错误) → **样例验证** (使用样例数据对程序进行验证输出与样例输出是否一致)

养成**先动脑，后动手**的习惯。清晰的逻辑思路，远比一开始就埋头敲错误的代码要高效得多。