语句构造

指针的构造技巧（7 条）

- 把指针操作独立在子程序中
- 使用指针之前对它进行检查
- 按照正确的顺序删除链表中的指针
- 不要吝惜于指针变量的使用，要专用
- 避免指针的强制类型转换
- 删除与释放指针需要事前检查，事后善后
- C++考虑使用智能指针

使用 if-then-else 时遵循的指导原则

- 首先编写正常的代码路径，再处理不常见情况
- 把正常情况的处理放在 if 后面而不要放在 else 后面
- 确保对于等量的分支是正确的
- 让 if 子句后面跟随一个有意义的语句
- 检查 else

使用布尔函数将判断过程独立出来

短路求值

避免深层嵌套的方法

- 重复判断一部分条件
- 转换成 if-then-else
- 转换成 case 语句
- 把深层嵌套的代码提取成单独的子程序
- 使用对象和多态派分

什么时候使用带退出循环

- 改写一个半循环为一个带退出循环

带退出循环的注意事项

- 把所有的退出条件放在一处
- 用注释来阐明操作意图
- 带退出的循环是单入单出的结构化控制结构，也是一种首选的循环控制

循环容易出现的错误

- 忽略或错误地对循环执行初始化
- 忽略了对累加变量或其他与循环有关的变量执行初始化
- 不正确的嵌套
- 不正确的循环终止
- 忽略或者错误地增加了循环变量的值
- 用不正确的循环下标访问数组元素

设计循环的理念

- 使循环尽量模块化

从三个方面考虑循环控制

- 如何进入循环
  - 只从一个位置进入循环
  - 把初始化代码紧放在循环前面
  - 用 while(true)表示无限循环
  - 在适当的情况下多使用 for 循环
  - 使用 while 循环更适用（控制部分较为复杂）的时候，不要使用 for 循环
- 处理好循环体
  - 避免空循环（原因很可能是把循环处理的代码和检测循环是否终止的代码写在同一行里）
  - 把循环内务操作要么放在循环的开始，要么放在循环的末尾
- 如何退出循环
  - 设法确认循环能够终止
  - 使循环终止条件看起来很明显
  - 不要为了终止循环而胡乱改动 for 循环的下标
  - 避免出现依赖于循环下标最终取值的代码
    - 可在循环体内某个适当的地方把这一最终取值赋给某个变量

结构化编程的三个组成部分

- 顺序
- 选择
- 迭代

使用防卫子句和错误处理代码来为正常路径铺路

使用递归的技巧

- 确认递归能够停止
- 使用安全计数器防止出现无穷递归
- 留心栈空间

表驱动的方法：

- 直接访问
- 索引访问
- 阶梯访问
  - CDF 和别名方法（二维）

GOTO 的优点和缺点

反对 GOTO 的论点：

- 含有 goto 的代码很难安排好格式
- 使用 goto 也会破坏编译器的优化特性
- 使用 goto 会使运行速度变慢，而且代码也更大

支持 GOTO 的观点

- 如果使用位置恰当，goto 可以减少重复的代码
- goto 在分配资源、使用资源后再释放资源的子程序里非常有用
- 在某些情况下，使用 goto 会让代码的运行速度更快，体积更小

代码改善

软件质量属性（两方面）

外在特性（运行时）(8 条)：

- 正确性
- 易用性
- 效率
- 可靠性
- 适应性
- 完整性
- 精确性
- 健壮性

内在特性（编码）（7 条）：

- 可维护性
- 灵活性
- 可移植性
- 可重用性

- 可读性
- 可测试性
- 可理解性

提高软件质量的方法（5 条）
- 明确质量管理目标
- 确定质量保证活动
- 测试策略
- 软件工程指南
- 技术检查

推荐技术阵容（4 条）
- 对所有的需求、架构以及系统关键部分的设计进行正式检查
- 建模或者创建模型
- 代码阅读或者检查
- 执行测试

协同构建包括哪些活动（5 点）
- 结对编程
- 正式检查
- 非正式技术复查
- 文档阅读
- 其他（如公开演示）

协同构建的优点（2 点）
- 有利于传授公司文化以及编程专业知识
- 集体所有权适用于所有形式的协同构建

结对编程的准则（9 点）
- 用编码规范来支持结对编程
- 不要让结对编程变成旁观
- 不要强迫在简单的问题上使用结对编程
- 有规律地对结对人员和分配的工作任务进行轮换
- 鼓励双方跟上对方的步伐
- 确认两个人都能够看到显示器
- 不要强迫程序员与自己关系紧张的人组队
- 避免新手组合
- 指定一个组长

详查和普查的区别（5 点）

- 详查表关注的是复查者过去所遇到的问题
- 详查专注于缺陷的检查，而非修正
- 复查人员要为详查会议预先准备，并带一份他们所发现的已知问题列表
- 高层管理人员不参加详查会议
- 每次详查所收集的数据都会被应用到以后的相差中，以便对详查进行改进

详查的一般步骤（7 点）

- 计划
- 概述
- 准备
- 详查会议
- 详查报告
- 返工
- 跟进

开发者测试的方法（5 点）

- 单元测试
- 组件测试
- 集成测试
- 回归测试
- 系统测试

结构化的基础测试

基本思想（3 点）

- 需要测试程序每条语句至少一次
- 要确保已经覆盖了所有的基础情况
- 以最小数量的测试用例覆盖所有路径

基本方法（1 点）

- 最简单的方法就是计算通过程序的路径，然后据此开发出能通过程序里每条路径的最少数量的测试用例

测试用例数的计算方法

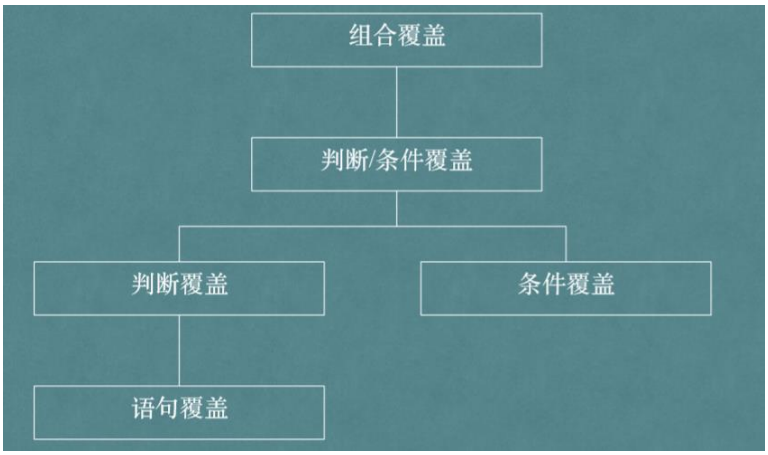
最简单的计算方法（3 点）

1. 对通过子程序的直路，开始的时候记 1
2. 遇到下面的每个关键字或者其等价物时加 1
    - if、while、repeat、for、and 以及 or

3. 遇到每个 case 语句就加 1，如果 case 语句没有缺省情况，则再加 1

测试覆盖率：功能点覆盖率和结构覆盖率（2 种）

- 功能点覆盖率大致用于表示软件已经实现的功能与软件需要实现的功能之间的比例关系
- 结构覆盖率包括（6 点）
  - 语句覆盖率
  - 判定覆盖率
  - 条件覆盖率
  - 判定/条件覆盖率
  - 组合覆盖率
  - 路径覆盖率



逻辑覆盖法（6 种）

- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定/条件覆盖
- 组合覆盖
- 路径覆盖

已定义-已使用数据流测试

- 不仅定义的地方要进行测试，使用的地方也要测试

等价类划分

猜测错误的常见情况：

- 边界值分析
- 典型的坏数据
  - 数据太少
  - 太多的数据
  - 错误的数据情况
  - 长度错误的数据
  - 未初始化的数据
- 几类好数据
  - 正常的情形
  - 最小的正常局面
  - 最大的正常局面

    - 与旧数据的兼容性
- 采用容易手工检查的测试用例

测试支持工具

- Diff 工具
- 测试数据生成器
- 符号调试器
- 系统干扰器
- 数据记录器
- 日志记录器