# 《软件构造》课程复习纲要

**参考教材《代码大全（第二版）》：以下简称CC2**

**复习所涉及的内容以PPT中出现的为准，具体细节参阅CC2中的解释。**

## 软件构造的概念

**软件构造不等于编码实现，软件构造是编码和调试，包括部分的详细设计和单元测试。**

1. **测试和调试的区别**

* **测试是为了发现错误**
* **调试是为了改成错误**

1. **软件开发包括哪些：**

* **问题定义**
* **需求分析**
* **实现计划**
* **总体设计**
* **详细设计**
* **编码实现**
* **系统集成**
* **单元测试**
* **系统测试**
* **校正性的维护**
* **功能强化**

1. **软件构造活动主要是其中哪些部分？**

**部分的详细设计，编码，和单元测试。**

**软件构造活动包括了：**

1. **验证基础工作已经完成，可以进行创建工作；**
2. **设计和编写子程序与模块**
3. **创立数据类型并命名变量**
4. **选择控制结构并组织语句块**
5. **找出并修正错误**
6. **评审其它小组的细节设计和代码，同时接受其它小组评审**
7. **通过仔细地格式化和征集意见改进编码**
8. **对分别完成的软件单元进行综合**
9. **调整编码使其更小、更快**
10. **软件构造的重要性**
    * **构造活动是开发软件的重要组成部分**
    * **构造活动在开发软件中处于枢纽地位**
    * **把主要精力集中于创建活动，可以极大地提高程序员的生产效率**
    * **创建活动的产品，源代码，往往是软件的唯一精确描述**
    * **创建活动是唯一一项必不可少的工作**
11. **隐喻（Software Metaphors）**

* **常见的软件隐喻 （简单了解）**

**软件书写：写代码（Writing Code）**

**软件播种：生成系统（Growing a System）**

**软件珍珠培植法：系统积累（System Accretion）**

**软件创建：建造软件（building software）**

**实用软件技术：智能工具箱（The Intellectual Toolbox）**

**防御性编程：防御式驾驶**

## 软件构造的前期准备工作

**准备工作的中心目标是降低风险**

**什么是软件构造的前期准备工作？**

**软件构造的前期准备工作是复查性的活动，明确前导性的工作结果有哪些，重新审视前导性工作是否完成 ，明确前导性工作完成的程度如何 。**

1. **软件构造的前期准备工作有哪些** 
   * **辨明软件项目的类型**
   * **明确问题定义**
   * **明确需求规约**
   * **明确软件架构的分个组成部分**

**软件项目的类型：常见有3种：商业系统，使命攸关的系统，性命攸关的系统。**

**商业系统倾向于使用高度迭代的开发方式，性命攸关的系统往往要求更加序列式的方法**

**序列式开发和迭代式开发：**

**序列式开发**

* + **使用瀑布式模型进行开发**
  + **适用于需求相对稳定的情况**

**迭代式开发**

* + **整个开发工作被组织为一系列的迭代过程。每一次迭代都包括了需求分析、设计、实现与测试。**
  + **适用于需求容易变化的情况**
* **倾向于序列式开发的情况** 
  + **需求相当稳定**
  + **设计相对简单、容易**
  + **后期改变需求、设计和编码的代价非常高**
* **倾向于迭代式开发的情况** 
  + **需求相对不稳定，或需要不断加深理解**
  + **设计复杂，具有挑战性**
  + **后期改变需求、设计和编码的代价比较低**

**问题定义指明项目要解决的问题是什么。**

**明确的问题定义是构造的第一项先决条件：在开始软件构造之前首先要check是否已经对系统要解决的问题做出了明确的陈述。**

**问题定义并不涉及任何可能的解决方案，应该用客户容易理解的语言编写，从客户的角度编写。**

**需求定义规定了软件系统应该做什么，在开始软件构造之前要check是否已经充分详尽地描述了系统所要做的事情。**

**下面是需求定义的核对表（了解）**

**核对表1：功能需求**

* **是否详细定义了系统的全部输入** 
  + **输入的来源、精度、取值范围、出现频率等**
* **是否详细定义了系统的全部输出** 
  + **输入的目的地、精度、取值范围、出现频率等**
* **是否详细定义了所有输出格式** 
  + **Web页面、报表、磁盘文件**
* **是否详细定义了所有硬件及软件的外部接口** 
  + **外部接口指该软件实体与外部硬件对接以及其它软件交互部分的接口**
* **是否详细定义了全部外部通信接口** 
  + **握手协议、纠错协议、通信协议等**
* **是否列出了用户想要做的全部事情**
* **是否详细定义了每个任务所用及得到的数据**

**核对表2：分功能需求**

* **非功能需求** 
  + **指软件产品在功能以外的服务质量（Quality of Service，QoS）方面的需求**
* **是否为必要操作定义了期望的相应时间** 
  + **在实时系统中是响应时间必要的**
* **是否详细定义了其它与计时有关的考虑** 
  + **处理时间、数据传输率、系统吞吐量**
* **是否详细定义了安全级别**
* **是否详细定义了可靠性** 
  + **包括软件失灵的后果、需要保护的重要信息、错误检测以及恢复策略**
* **是否详细定义了内存、硬盘的使用要求**
* **是否详细定义了系统的可维护性** 
  + **功能的变更、操作环境的变更和接口的变更**
* **是否包括对“成功”、“失败”的定义**

**其他方面的核对表：**

* **需求的质量** 
  + **需求定义自身所表现出来的质量**
  + **需求的清晰性、一致性、可测试性等**
* **需求的完备性** 
  + **需求是否可以覆盖问题定义所描述的信息**
  + **是否不包含不能实现的需求**

**软件架构（Software Architecture）是软件设计的高层部分，用于支撑更加细节的设计的框架，在开始软件构造之前要check是否已经在整个系统范围内定义了相应的框架结构。**

**软件架构是构建计算机软件实践的基础**

* + **软件架构定义了一张用于描述整个系统各个方面的草图**
  + **是一个软件系统从整体到部分的最高层次的划分**
  + **建造一个系统所作出的最高层次的、以后难以更改的，商业的和技术的决定**

**软件架构的两个要素：元件划分和设计决定**

**架构的主要内容：程序的组织结构，定义主要的类，数据设计，业务规则，安全性，国际化/本地化，错误处理，容错性， 变更策略。**

**优秀架构的特点**

* + **描述了开发过程中的多种变更方式**
  + **描述了主要决策的动机**
  + **保持较高的编程语言无关性**
  + **指出有风险的区域**
* **包含对系统的多个视图**

## 构造技术

1. **模块化设计 （简单了解相关概念）**

**这一部分的复习参阅CC2书上的第5章**

**模块的语言支持包括：数据，数据类型 ，数据操作 ，公共和局部操作**

**模块需要在局部，在模块中及在全局中三个层次上区分可见性**

1. **模块和子程序的区别** 
   * **子程序是具有一定功能的，可以调用的函数或过程**
   * **模块是指数据及作用于数据的子程序的集合，或者**
   * **模块也可能是指，可以提供一系列互相联系功能的子程序集合，而这些子程序之间不一定有公共的数据**
2. **模块化：内聚性与耦合性（重要）**

**模块独立程度的衡量标准:耦合性（ Coupling ）和内聚性（Cohesion）**

**模块的内聚性表示了在一个模块中，各种操作及数据之间互相联系的紧密程度，其强弱程度用“强度”表示，应尽量提高内聚程度。**

**模块内聚性分类：偶然内聚，逻辑内聚，时间内聚（都为低内聚）**

**子程序内聚性：**

**可取内聚性：功能内聚（最高），顺序内聚（较高）通讯内聚（中）临时内聚（低）**

**不可取内聚性：过程内聚性，逻辑内聚性，偶然内聚性**

**功能内聚性：当程序执行一项并且仅仅是一项工作时，属于功能内聚**

**顺序内聚性：子程序内包含需要按特定顺序进行的、逐步分享数据而不形成一个完整功能的操作。**

**通讯内聚性：在一个子程序中，两个操作只是使用相同数据，而不存在其它任何联系时产生的。**

**临时内聚性：因为同时执行的原因才被放入同一个子程序里，这时产生内聚性。**

**过程内聚性：当子程序中操作是按某一特定顺序进行的，就是过程内聚性，与顺序内聚性的区别在于：过程内聚性中的操作使用的并不是相同数据。**

**逻辑内聚性：当一个子程序中同时含有几个操作，而其中一个操作又被传进来的控制标志所选择时，就产生了逻辑内聚性，其内部操作仅仅是因为控制流，或者说“逻辑”的原因才联系到一起。**

**偶然内聚性：当同一个子程序中的操作之间无任何联系时，为偶然内聚性，也称为“无内聚性”**

**模块耦合性：表示相互作用的模块之间的互连程度，应尽量降低模块之间的耦合性。**

**耦合分类：**

* **无任何连接：两个模块中的每一个都能独立地工作而不需要另一个的存在（最低耦合）。**
* **数据耦合：两个模块彼此通过参数交换信息，且交换的仅仅是数据（低耦合）。**
* **控制耦合：两个模块之间传递的信息有控制成分（中耦合）。**
* **公共环境耦合：两个或多个模块通过公共环境相互作用** 
  + **一个存数据，一个取数据（低耦合）**
  + **都存取数据（低--中之间）**
* **内容耦合** 
  + **一个模块访问另一个模块的内部数据**
  + **两个模块有一部分程序代码重叠**

**常见OO耦合：**

* **简单数据参数耦合** 
  + **两个对象之间通过参数传递数据，并且为简单数据类型**
* **简单对象耦合** 
  + **一个对象实例化另一个对象所形成的耦合关系**
* **对象参数耦合** 
  + **两个对象之间通过对象参数传递数据**
* **语义上的耦合**

**信息隐蔽，又称为“封装”，使可见部分与内部不可见部分相互隔离，分别称为模块的内部信息和公开信息或接口。**

**常见需要隐含的信息：容易被改动的区域，复杂的数据，复杂的逻辑，在编程语言层次上的操作**

**容易被改变的区域：对硬件有依赖的地方，输入和输出，状态变量，数据规模限制，商业限制。**

1. **结构化设计vs.面向对象设计**

**从模块的角度理解结构化设计和OO设计的区别**

* + **面向对象设计中的模块与结构化设计中模块的含义是一致的** 
    - **数据和功能的封装**
    - **满足信息隐蔽的要求**
    - **追求高内聚低耦合**
  + **结构化设计中模块和模块之间的关系，被紧紧局限于信息流（结构化设计是面向功能）**
  + **面向对象设计方法，充分挖掘了“关系”的表达方式，可以尽可能的将事物之间复杂的关系予以体现**

1. **抽象数据类型（ADT）**

**了解基本概念，能够构造出简单的ADT**

**ADT是指一组数据以及对这些数据所进行的操作的集合**

**例： Stack**

**Initialize stack**

**Push item onto stack**

**Pop item from stack**

**Read top of stack**

**ADT的实现：在面向对象环境中无需ADT多份实例的细节，在非面向对象环境中要手工支持多实例技术：添加用来创建和删除实例的服务操作，重新调整ADT的其它服务操作**

1. **ADT与类的区别**

**体会ADT和类在概念上有什么区别。**

* + **从数据的角度来看类和ADT没有区别**
  + **从OO的角度来看，类还涉及到面向对象中的概念**
* **类是以面向对象概念做支撑的ADT**
  + **封装的概念**
  + **继承的概念**
  + **多态的概念**
* **ADT中的封装与类的封装有本质区别** 
  + **ADT是弱封装**
  + **类通过可见性实现真正意义的封装（即模块化的黑盒）**

**给你一个ADT，构造出相应的类（Java或C++）**

1. **良好的类接口**

**类接口和接口的区别**

* + **类接口指类所公开的那部分操作**
  + **接口指仅有操作的声明而不包含实现部分**

**良好的类接口需要:一致的抽象,良好的封装**

**给你不良好的类接口，能够指出其中的问题，并加以改进**

**（难度限于书中例子的难度）**

1. **包含和继承的选择 （了解）**

**类之间的包含（Containment）关系：描述了一个类拥有另外一个类的关系，包含可以理解为“有一个……”的关系**

**类之间的继承（Inheritance）关系：描述了一个类是另一个类的特化的关系，继承可以理解为“是一个……”的关系**

**继承关系中的Liskov替换原则（LSP）：派生类必须能够通过基类的接口被使用，且使用者无需了解两者之间的差异**

## 防御式编程

1. **防御式编程的主要思想**

**防御式编程时防御式设计的一种表现形式，它是为了保证对程序的不可预见的使用不会造成之后程序功能的破坏。**

**防御式编程的思想可以看成是为了尽量消除或降低墨菲法则带来的影响。**

1. **墨菲法则 （了解）**

**墨菲法则是指：“任何有可能出错的事终将出错”**

1. **断言部分（重要）**

**基本概念、使用方式**

**两种定义：1.断言是一种布尔表达式，用于在程序中表明该处所要满足的条件2.断言是指在开发期间使用的、让程序进行自检的代码。**

**使用方式：一般断言通常含有两个参数：一个描述假设为真的情况的布尔表达式，一个断言为假时需要显示的信息。**

1. **建立自己的断言机制**

**能够使用Java、c++、c#等任意一种语言建立自己的断言机制**

**C++示例：一个实现断言的宏**

**#define ASSERT（condition，message）**

**If(!(condition))(**

**logError(“Assertion failed:”,#condition,message);**

**exit(Exit\_FAILURE);**

**}**

1. **使用断言的指导原则/建议**

* **用错误处理代码来处理预期会发生地状况，用断言来处理绝不应该发生地状况**
* **避免把需要执行的代码放到断言中**
* **用断言来注解并验证前条件和后条件**
* **对于高健壮性的代码，应该先使用断言再处理错误。**

1. **错误处理和断言**

**了解两者的主要区别**

**错误处理代码是用来检查不太可能经常发生地非正常情况，如，检查有害的输入数据，错误处理属于程序运行的“正常”情况**

**断言是用来检查永远不该发生地情况，用于检查代码中的bug，触发了断言则应该修改程序的源代码并重新编译，然后发布软件的新版本。**

1. **避免把需要执行的代码放到断言中**

**给出代码示例，能够指出此类问题**

**断言中应该只放入逻辑判断，**

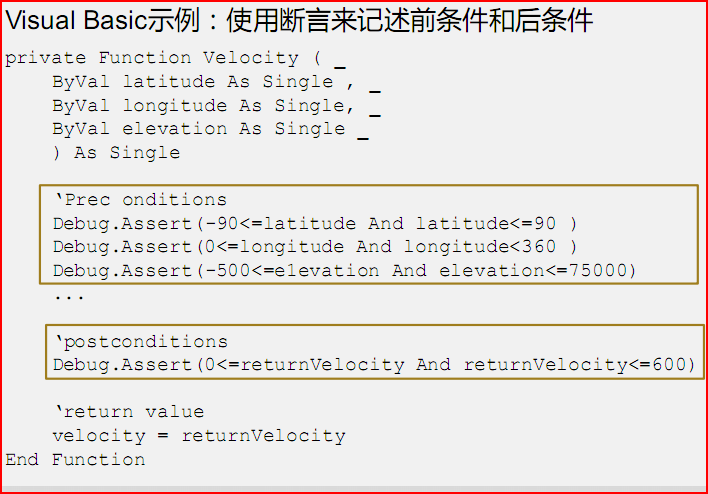
**错误代码示例：Visual Basic ：**

**Debug.Assert(PerformAction())**

**`Couldn’t perform action**

1. **用断言来验证前置和后置条件**

**给出代码示例，能够指出其中断言属于哪种应用**

****

1. **错误处理技术 （了解）**

**返回中立值（如0），换用下一个正确的数据，返回和前次相同的数据，混用最近的合法值，把警告信息记录到日志文件中，用语言内建的异常处理机制抛出一个异常，返回一个错误码，调用错误处理子程序或对象，在局部处理错误。**

1. **异常处理机制（重要）**
2. **异常的基本结构**

**子程序使用throw抛出一个异常对象，再被调用链上层其他子程序的try-catch语句捕获。**

1. **异常处理机制所涉及的要素**

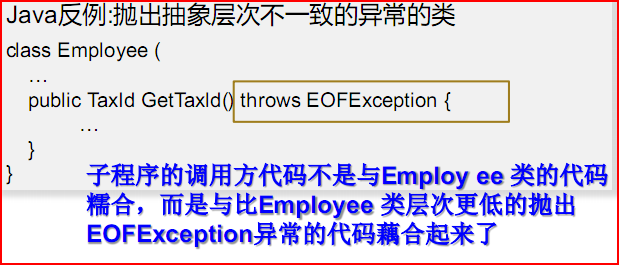
**异常对象，对异常对象的抛出，对所抛出异常的捕捉**

1. **finally一般被用来进行清理工作**

**最常见的就是关闭流，关闭连接，释放或销毁资源等。**

1. **改进使用不当的异常情况**

**能够识别出代码中使用不当的异常（如，ppt P40，41 ）**

****

**避免使用空catch语句。**

1. **隔离可以划分错误处理和断言**

**理解 通过隔离代码将错误处理和断言两种技术合理应用到自己的系统中。**

**隔离部分包含了“脏数据”，隔离部分的程序应使用错误处理技术，在那里对数据做的任何假定都是不安全的。通过隔离部分之后的是“干净数据”，隔栏内部的程序里就应使用断言技术因为传进来的数据应该已在通过隔栏时被清理过了。**

1. **产品版和开发版**

**了解两者的区别**

* + **产品级的软件要求能够快速地运行，而开发中的软件则允许运行缓慢**
  + **产品级的软件要节约使用资源，而开发中的软件在使用资源时可以比较奢侈**
  + **产品级的软件不应向用户暴露可能引起危险的操作，而开发中的软件则可以提供一些额外的、没有安全网的操作**

1. **进攻式编程**

**了解什么是进攻式编程**

**进攻式编程是主动暴露可能出现错误的态度，在开发阶段让它显现出来，而在产品代码运行时让它能够自我恢复**

**常用的进攻式方法 ：**

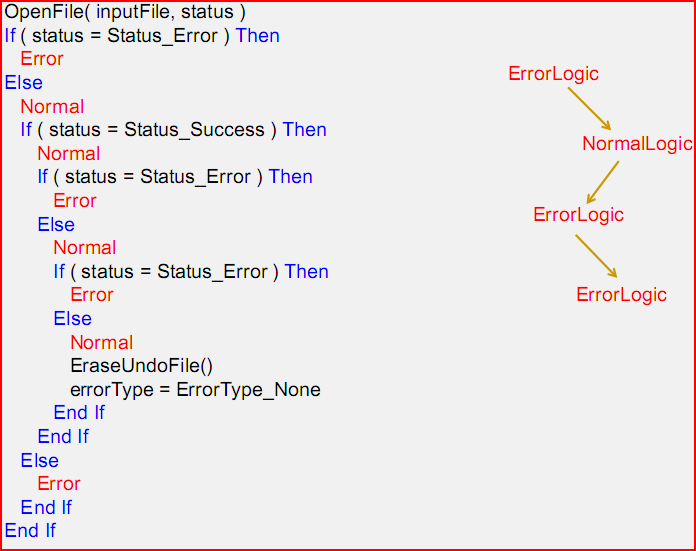
**确保断言语句使程序终止运行；完全填充分配到的所有内存 ；完全填充己分配到的所有文件或流 ；确保每一个case 语句中的default分支或else 分支都能产生严重错误（如终止程序）；在删除一个对象之前把它填满垃圾数据**

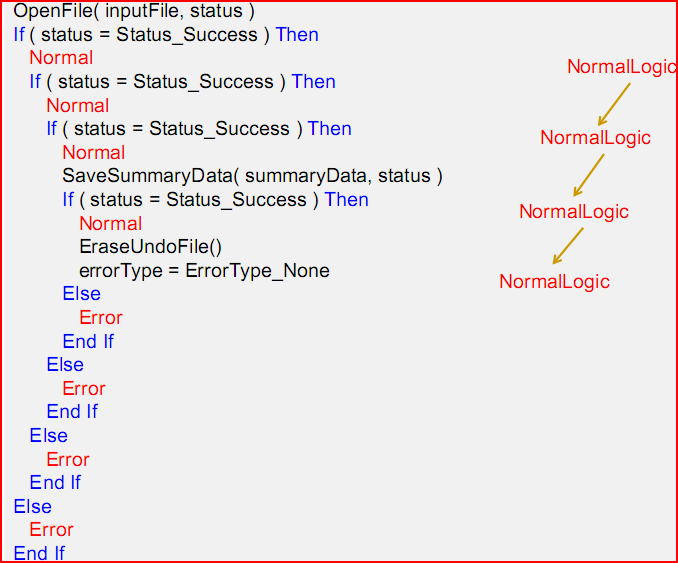
## 语句编写的技巧

1. **条件语句 （重要）**
2. **优先考虑正常代码路径**

**能够根据此原则对代码进行优化调整（难度限于CC2和PPT）**

**调整策略：正常的路径一致地写在前面，即if语句后面，所有错误的情况都写在后面，即else语句后。**

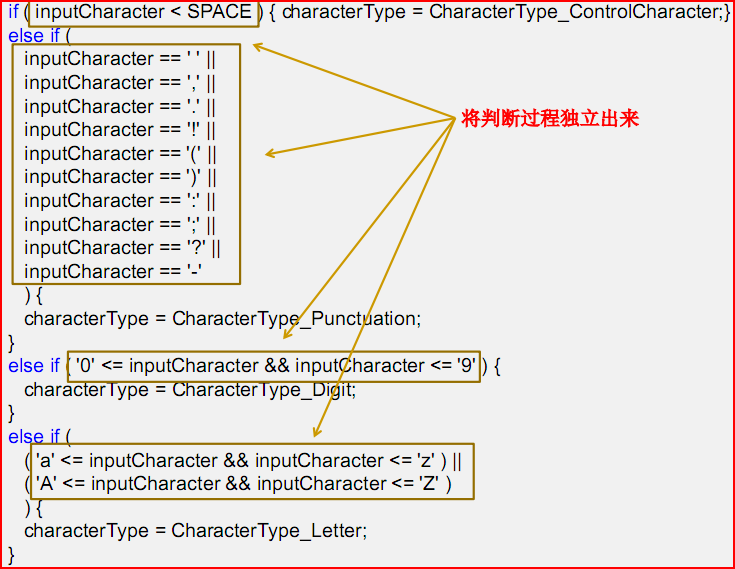
**例：**

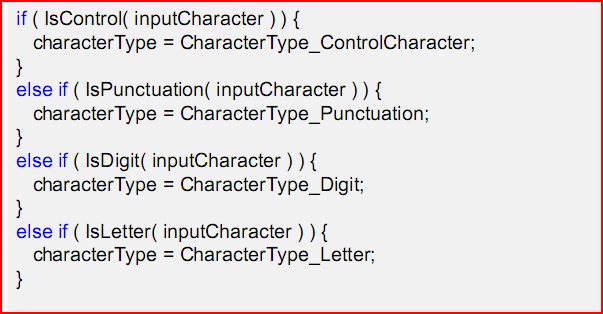
****

1. **利用布尔函数调用简化复杂的检测**

**能够根据此原则对代码进行优化调整（难度限于CC2和PPT）**

**复杂的判断情况会影响代码的可读性，将复杂的判断从代码中隔离开，通过布尔函数封装判断过程。**

****

****

1. **循环语句（重要）（难度限于CC2和PPT）**
2. **循环的种类**

**了解常见循环语句分类各自的特点**

**计数循环：执行的总次数是一定得**

**连续求值的循环：预先不知道执行多少次，在每次送达时检查是否应该结束**

**无限循环：一旦启动就会一直执行下去**

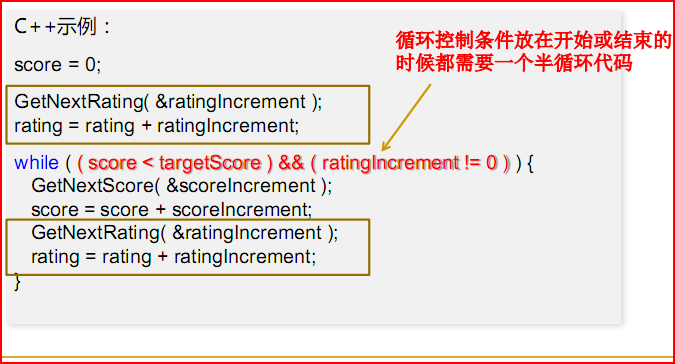
**迭代器循环：对容器类里的每个元素执行一次操作。**

1. **带退出的循环和半循环**

**能够使用带退出循环对带半循环的代码进行改写**

**带退出的循环就是终止条件出现在循环中间而不是开始或末尾的循环。**

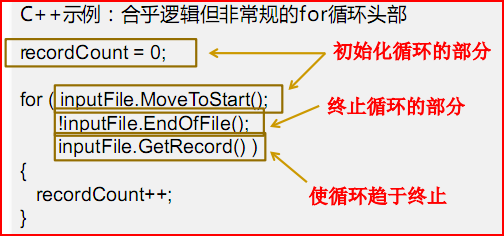
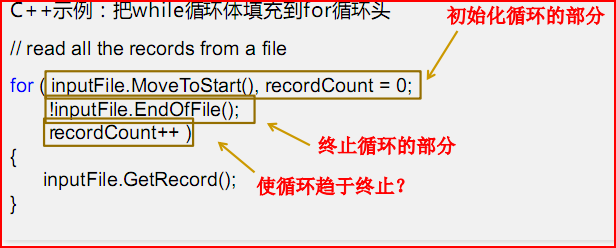
**所谓半循环的代码就是为了完成整个循环而设置在整个循环体之外的代码。**

****

****

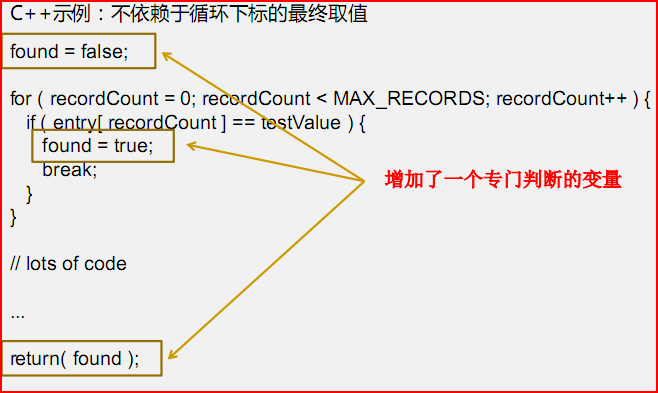
1. **while循环和for循环**

**能够对while循环和for循环代码片段进行相互改写**

****

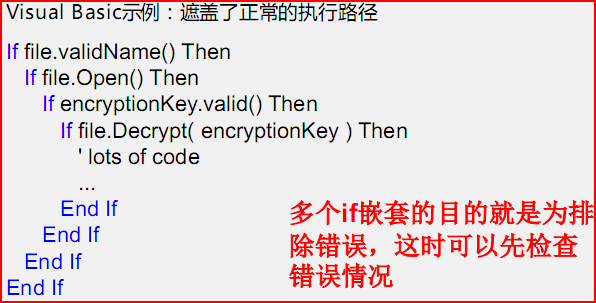
1. **避免依赖于循环下标最终取值**

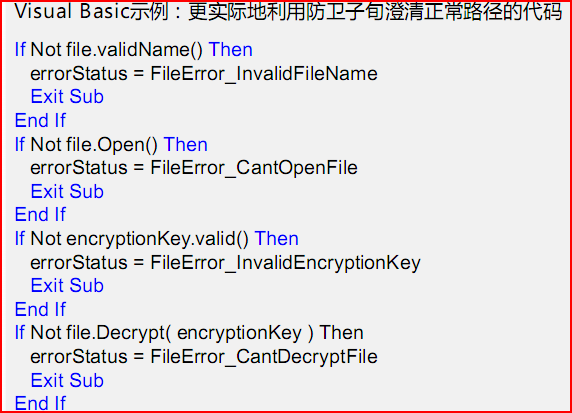
**能够对依赖循环下标最终取值的代码进行改进**

****

1. **用防卫子句( guard clause) ( 早返回或早退出) 来简化复杂的错误处理**

**能够用防卫子句对代码进行改进**

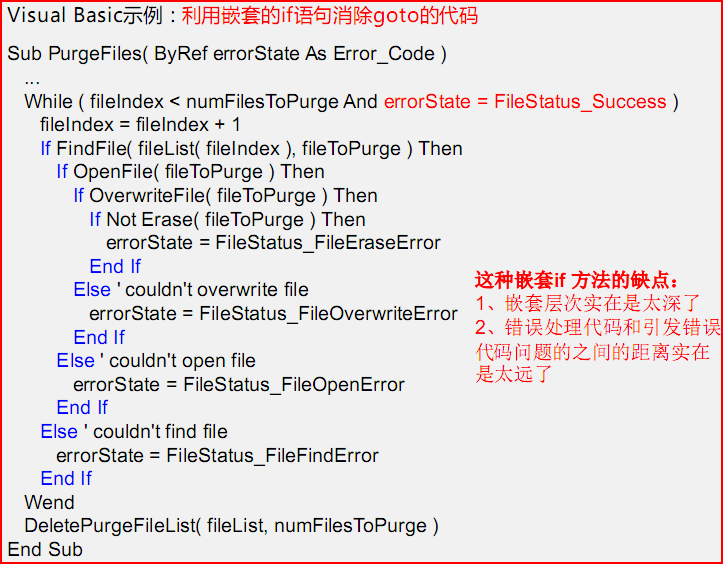
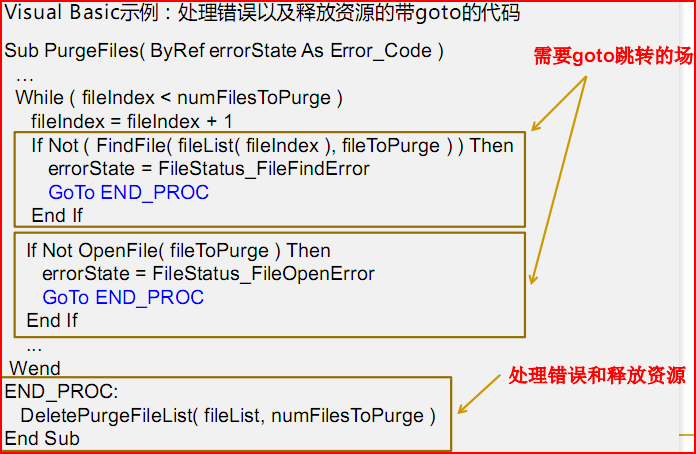
****

****

1. **无条件跳转语句：GOTO**

**了解goto机制的背景和应用原则**

**goto是无条件的跳转，荷兰科学家Edsger W.Dijkstra最早提出goto有害，**

****

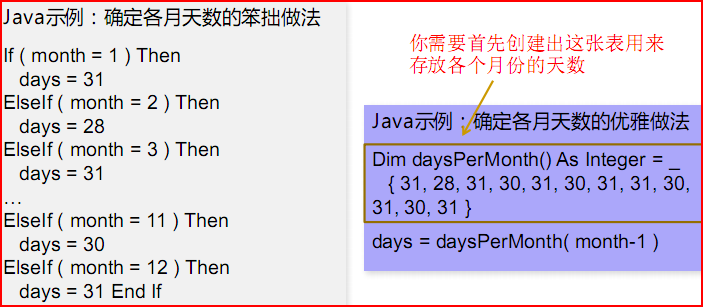
1. **表驱动法（重要）（难度限于CC2和PPT）**

**掌握直接访问、索引访问和阶梯访问的概念，能够：**

* 1. **判断代码中使用的表驱动技术属于哪一种**
  2. **理解代码中“表”的定义和使用**

**其中重点掌握直接访问表、索引访问表**

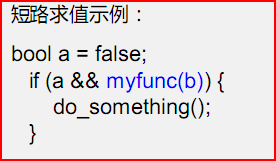
**直接访问是指通过索引值（如下标）可以直接从表中找到对应的条目**

****

**索引访问表就是现用一个基本类型的数据从一张索引表中查出一个键值，然后再用这一键值查出你感兴趣的主数据（间接访问）**

**阶梯结构的基本想法：通过确定每项命中的阶梯层次确定其归类**

1. **短路求值**

**了解短路求值的概念**

## 代码改善

1. **软件质量的特性**
2. **了解外在特性和内在特性**

**外在特性：指的是该产品的用户所能够感受到得部分；质量的外在特性是用户关心的唯一软件特性，指软件是够容易使用和正确运行等**

**内在特性：软件内部结构、组织所体现出来的特性，软件在代码层次上具有的特性。**

**外在特性包括：正确性，可用性，效率，可靠性，适应性，完整性，精确性，健壮性**

**内在特性包括：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**

1. **常见的测试**

**了解单元测试、组件测试、集成测试、回归测试以及系统测试的概念**

**单元测试：将一个程序员或者一个开发团队所编写的，一个完整的类、子程序或者小程序，从完整的系统中隔离出来进行测试。**

**组件测试：将一个类、包、小程序或者其他程序元素，从一个更加完整的系统中隔离出来进行测试**

**集成测试：是对两个或更多的类、包、组件或者子系统进行的联合测试，这些组件由多个程序员或者开发团队所创建**

**回归测试：是指重复执行以前的测试用例，以便在原先通过了相同测试集合的软件中查找缺陷**

**系统测试：在最终的配置下（包括同其他软硬件系统的集成）运行整个软件**

1. **测试的特殊性**

**理解所谓测试的特殊性**

**测试的目标与其他开发活动背道而驰，测试的目的是找出错误**

**测试永远不可能彻底证明程序中没有错误**

**测试本身并不能改善软件的质量**

**测试时要求你假设会在代码里面找到错误**

1. **不完整的测试**

**理解为什么不可能穷尽测试以及测试的基本出发点**

**测试的基本出发点：**

**由于进行完全测试实际上是不可能的，因此测试的窍门就在于选择那些最有可能找到错误的测试用例**

**当规划测试的时候，要去除那些不会告诉你任何新情况的测试用例**

**使用不同的方法来有效地覆盖程序的基本情况**

1. **结构化的基础测试（重要）**

**掌握基本概念以及基础测试用例最少数量的简单计算方法，能够根据代码设计相应的测试用例**

**基本思想：需要去测试程序中的每一条语句至少一次；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了所有路径。**

**基本方法：最简单的方法就是算一算有多少条通过程序的路径，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。**

**基础测试用例最少数量的简单计算方法：**

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,or**

1. **遇到每一个case语句加1，如果case语句没有缺省情况，则再加1**