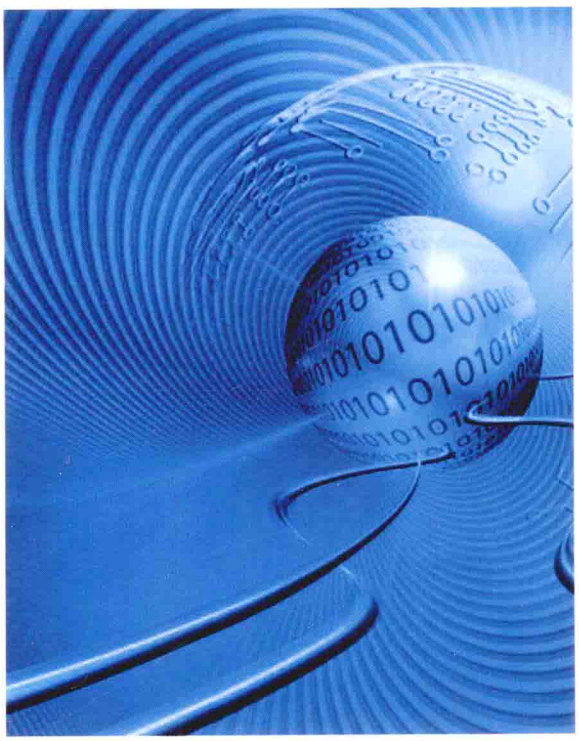
高等学校计算机应用规划教材

# 数据结构实验指导教程(C语言版)

- 预备知识
- 线性表
- 栈与队列
- 字符串
- 数组
- 二叉树与树
- 图
- 查找表
- 排序
- 课程设计



杨海军　马　彦　叶燕文　主　编

清华大学出版社

高等学校计算机应用规划教材

# 数据结构实验指导教程

## (C 语言版)

杨海军　马　彦　叶燕文　主编





清华大学出版社

## 内 容 简 介

本教程中预备知识，介绍项目中实验环境、程序组织方式和管理方法，讲解程序及算法的效率估算方法和分析技巧，描述一般测试技术和调试方法，以及测试数据与测试用例的设计安排技巧；基础实验，首先安排重要的验证性实验，然后设计基于基本数据结构的简单应用实验；综合实验(即课程设计)，介绍在一个项目中选择和使用多种基本数据结构的依据和方法，讲解如何有效地将它们融合在一起解决实际的复杂应用问题。

本教程在内容选取及编排顺序上，与严蔚敏老师编著的《数据结构(C 语言版)》(978-7-302-14751-0，清华大学出版社出版)保持一致，可作为高等院校计算机及相关专业数据结构课程的实验教材。



**图书在版编目(CIP)数据**

数据结构实验指导教程：C 语言版 / 杨海军，马彦，叶燕文　主编. —北京：清华大学出版社，2014
(高等学校计算机应用规划教材)

ISBN 978-7-302-36259-3

Ⅰ. ①数…　Ⅱ. ①杨…　②马…　③叶…　Ⅲ. ①数据结构—高等学校—教材　②C 语言—程序设计—高等学校—教材　Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字(2014)第 076280 号

**责任编辑**：王　定　程　琪
**装帧设计**：牛艳敏
**责任校对**：邱晓玉
**责任印制**：何　芊

**出版发行**：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn,010-62794504

**印 装 者**：三河市吉祥印务有限公司
**经　　销**：全国新华书店
**开　　本**：185mm×260mm　　**印　张**：20　　**字　数**：462 千字
**版　　次**：2014 年 9 月第 1 版　　**印　次**：2014 年 9 月第1 次印刷
**印　　数**：1～2000
**定　　价**：35.00 元

产品编号：059075-01

# 前　言

数据结构是计算机程序设计的重要基础，如何培养学生的实际动手能力，从而解决具体问题就是本课程的基本任务之一。我们在实际教学中发现，虽然学生对数据结构基本概念及基础操作有一定认识，但仍然难以独立设计测试环境和测试用例，在面对具体应用问题时，缺乏选择适当的数据结构及实现算法的能力。我们认为，解决以上问题是改进数据结构教学中存在的“难教难学”状况的重要途径，为此，编写了这本《数据结构实验指导教程》。

本实验教程通常理论及实验教学内容以“基本数据结构—实现”和“问题识别—设计实现”的层次进行组织。为了降低学习曲线，导入了软件设计的基本概念，安排了测试环境、测试用例设计的基础内容，同时在基础实验与综合应用实验间安排了简单应用实验，以实现从概念到简单应用再到复杂应用的平滑过渡。

简单应用实验选择比较单一的实际问题，分析其逻辑结构，然后考虑在计算机环境下如何表达其结构，再利用现有四种基本数据结构中的一种算法来实现和完成。综合应用实验一般较复杂，需要利用多种基本数据结构来完成。

基础实验部分对应“基本数据结构—实现”层次，用以深化数据结构基本概念及其机器实现；简单应用实验部分和综合应用实验部分对应“问题识别—设计实现”层次，先识别具体问题中包含哪些基本数据结构，再根据前面的实现来完成设计。

本书从内容上共分三部分。

- 预备知识：对项目中实验环境、程序组织方式和管理方法有较深认识，需要理解程序及算法的效率估算方法和分析技巧，掌握一般测试技术和调试方法，尤其是测试数据及测试用例的设计安排。
- 基础实验：紧贴数据结构要点，以及基于基本数据结构的简单应用。目的在于使学生能面对问题，识别基本数据结构，会编程应用已有存储结构和算法实现。
- 综合实验(或课程设计)：介绍在一个项目中使用多种基本数据结构的方法，注重选择基本数据结构的依据，以及如何有效地将它们融合在一起。

本书第1章由杨海军编写，第2、4、6、8章由马彦编写，第3、5、7、9、10章由叶燕文编写。杨海军负责全书的规划和统稿，马彦、叶燕文负责全书的编排。

本书在编写过程中参考了诸多同行的文章和著作(见书后所附参考文献)，在此一并致谢。由于编者的知识水平有限，书中疏漏之处在所难免，恳请专家和读者批评指正。

编者

2014年5月

# 目 录

# 第1章 预 备 知 识

了解软件开发的基本过程和步骤，掌握软件设计的基本原理，熟悉高级语言编程工具以及测试调试的主要概念和方法，不仅能够有效地表示和实现数据结构及相关算法，也是完成本教程实验的重要保证。

## 1.1 软件开发过程与设计原理

### 1.1.1 软件开发的一般步骤

通常依据生命周期(Life Cycle)理论将软件开发步骤分为需求分析、软件设计、编码实现、测试及维护五个阶段。软件生命周期是指这样一个过程：从用户需求开始，经过开发、交付使用，在使用中不断地增补修订，直至软件报废。需求分析阶段的主要任务是准确地确定“软件系统必须做什么”，及确定软件系统必须具备哪些功能。概要设计就是决定软件由哪些模块组成、每个模块的功能以及模块间的调用关系和参数传递情况；同时决定该软件要存储的数据、这些数据的组成和相互关系，即设计该软件的总体数据结构或数据库结构。利用程序流程图、伪码等工具，将概要设计阶段确定每个模块的抽象功能转换为精确的、结构化的过程描述是详细设计阶段的主要任务。编码阶段就是把每个模块的控制结构转换成计算机可接受的程序代码，即写成以某特定程序设计语言表示的“源程序”。测试是保证软件质量的重要手段，其主要方式是利用测试用例检验软件的各个组成部分是否满足规格要求。一般的，将测试分为模块测试、集成测试和确认测试。软件正式运行后，便进入软件维护阶段，它可以持续几年甚至几十年，它是软件生存期中时间最长的阶段。

### 1.1.2 软件设计的基本原理

模块(Module)是构成程序的基本构件，是基本的设计元素。通常使用软件结构概念表示程序是由哪些模块构成的，以及这些模块相互间的关系。软件结构以层次表示程序的系统结构，即一种控制的层次体系，表示了软件元素(模块)之间的关系，如调用关系、包含关系、从属关系和嵌套关系等，并不表示软件执行的具体过程。

下面介绍几个与软件结构相关的概念。

#### 1. 模块化

模块是由边界元素限定的相邻程序元素的序列，且有一个总体标识符代表它，如过程、函数、子程序、宏等。

模块化(Modularity)是将系统划分为若干个模块，每个模块完成一个子功能。模块化的目的是将系统“分而治之”，因此能够降低问题的复杂性，使软件结构清晰，易修改、易阅读、易理解，因而也有助于提高软件的可靠性，同时使得系统各个部分可以并行开发，提高软件的生产率。但并非模块分得越小越好，原因是随着模块规模变小，模块的数量会增多，这样会引起模块之间接口的复杂度和工作量增加。

提高模块质量在于提高模块独立性。模块独立性的准则用耦合性和内聚性来衡量。进行系统模块划分时尽量做到高内聚、低耦合，即要使模块的内部联系尽可能地强，而模块间的外部联系尽可能地弱，这样就尽可能地提高模块的相对独立性。C 语言开发中，一个模块对应一个函数或宏，也就是说采用函数和宏实现模块。

### 2. 软件结构的描述

通常采用软件结构概念来描述软件的组成模块、各模块功能以及模块间通信协调的方法和方式，反映了软件的静态特性。软件结构决定了整个系统的结构，也确定了系统的质量。软件结构图、层次图是描述软件结构的常用工具。

(1) 软件结构图

软件结构图是精确描述软件结构的图形表示方法。它以特定的符号表示模块、模块间的调用关系和模块间信息的传递。软件结构图的主要元素有模块、调用和数据。

- 模块：用矩形框表示，框中写有模块的名字，说明模块的功能。模块是程序对象有名字的集合。如图 1-1 中的“计算通信费用”“计算市话费”等。
- 调用：从一个模块指向另一个模块的箭头表示前一模块对后一模块的调用，一般是上层调用下层。在图 1-1 中，在计算通信费用模块中调用计算市话费、计算国内长途费和计算国际长途费三个模块。
- 数据：调用箭头边上的短箭头表示调用时从一个模块传送给另一模块的数据。通常在短箭头附近应注有信息的名字，如图 1-1 所示。

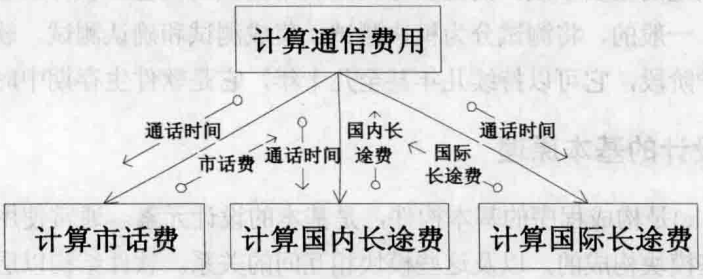



图 1-1　软件结构图示例

数据又分为数据信息和控制信息两类，如果要区分这两类信息，通常可用尾端带有空心圆的短箭头表示数据信息，用尾端带有实心圆的短箭头表示控制信息。有的结构图对这两种信息不加以区别，一律用注有信息名的短箭头来表示。

(2) 层次图

层次图用来描绘软件的层次结构，层次图中的一个矩形框代表一个模块，方框间的连

线表示调用关系，如图 1-2 所示。层次图很适于在自顶向下设计软件的过程中使用。层次图中的主要元素有模块和调用。

- 模块：用矩形框表示，框中写有模块的名字，说明模块的功能。
- 调用：方框间的连线，表示模块间的调用关系。

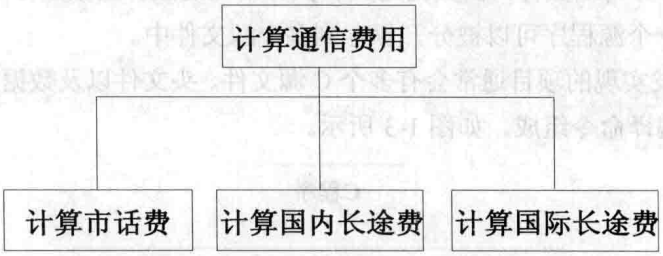



图 1-2 软件层次图表示

### 3. 详细设计与表示方式

概要设计用以确定软件的组成模块、各模块功能性以及彼此间接口，但还尚未仔细考虑模块具体的实现算法和数据结构等要素，这是详细设计的主要任务。为了把概要设计期间简明、无歧义的一般处理步骤的描述转换成为准确的、结构过程描述，需要使用设计描述工具。这些描述工具应当表现出控制的流程、处理功能、数据的组织以及实现的细节，可分为以下几种：

(1) 图形工具。把过程的细节表示成一个“图”的组成部分，在这个图上，逻辑构造用具体的图形来表示。大家熟知的程序流程图(也称为框图)及 N-S 盒图就属于该类型。

(2) 列表工具。用一个表来表示过程的细节，这个表列出了各种操作及其相应的条件。换句话说，描述了输入、处理和输出信息。用以描述复杂逻辑的决策表方法是典型代表。

(3) 语言工具。用类语言来表示过程的细节，这种类语言很接近于编程语言。例如，类 C 语言伪码就是一个典型代表。该伪码主要语法是 C 语言的控制结构，同时借用了 C 语言某些编程语言元素的简写符号，在实际使用时，允许设计人员可以随意增加伪码的详细程度，特别是针对可能产生二义性的地方。

## 1.2 C 语言程序的组织与管理

在大多数 C 语言教材中，教学例程功能单一，源程序规模很小，超过 50 行的就算大程序了。即使这样的例程，通常也仅由一个源文件和头文件组成，且源文件中函数也屈指可数。但实际应用中的项目规模相对较大，其中涉及资源较多，如何有效地进行管理是开发工具比较关注的焦点。目前主要的 C 编译器均提供项目管理功能实现较大程序的资源和源文件间依赖管理，用以提高开发效率。利用这种功能，开发大程序的工作将更加方便。C 语言的源程序是由函数、预处理命令和编译指令构成的集合。

### 1.2.1　C 语言程序的构成

C 语言中源程序的基本组织单位是函数，程序功能由多个用户函数和系统函数来完成。一个函数由两部分组成，一个是函数说明部分，包括函数名、函数类型、函数属性及函数参数情况；另一部分为函数体，由数据定义和执行语句组成。函数是 C 语言实现程序模块化的主要机制。一个源程序可以被分开保存到多个源文件中。

用 C 语言开发实现的项目通常会有多个 C 源文件、头文件以及数据文件，每一源文件由多个函数及预编译命令组成，如图 1-3 所示。

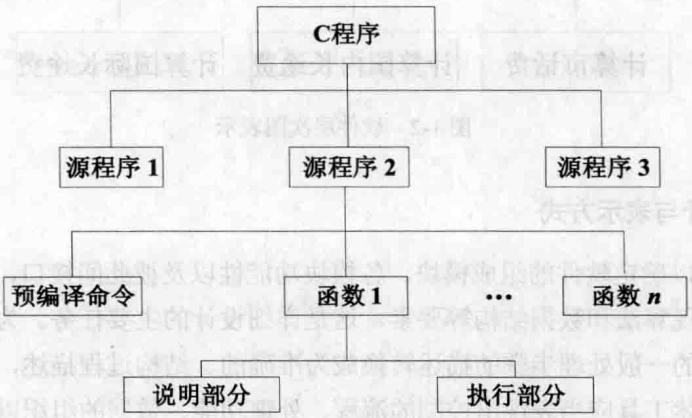



图 1-3　C 语言结构图示例

### 1.2.2　较大程序的实现方法和项目管理

实际开发活动中，需要建立项目组，由项目经理进行管理。在编码阶段，根据设计要求会在程序员间进行分工，各程序员编写的源代码放在各自的源文件中，分别进行编译和调试，最后将每个程序员的源文件组合起来，形成整个软件。项目中源文件间存在着一种依赖关系，这是因为一个源文件中的函数使用了另一源文件中定义的程序对象(如外部变量、函数、类型等)，习惯上将使用某程序对象的称为客户，而被使用的某程序对象则为服务提供者。这种依赖关系导致了对服务提供者的修改必然引起其他源文件中的客户代码也要做相应修改的后果。

### 1.2.3　程序的生成与调试

目前主要的算法语言除了拥有基本的编译器外，还集成了编辑器、调试器、函数库、联机文档及项目管理器等工具，形成了高效的软件开发环境。程序员工作流程一般经过以下步骤反复进行：

(1) 编辑程序。在编辑器中完成源程序的编写。

(2) 编译。使用编译器对源程序进行词法、语法分析的编译，若无错误则产生目标代码，否则将发现的所有错误列表出来供调试。

(3) 调试，改正所有语法错误。对于较复杂的程序，要利用调试器及调试技术进行跟

踪调试。

调试是指找到出错的原因与位置并纠错，包括修正源程序直到软件正确为止。虽然调试应该而且可以是一个有序过程，但是，目前它在很大程度上仍然是一项技巧，对经验要求较高，需要花费大量时间精力。软件测试结果往往体现了软件错误的症状，也就是说，软件错误的外部表现和它的内在原因之间可能并没有明显的联系。调试就是把症状和原因联系起来的尚未被人深入认识的智力过程。调试过程从执行一个测试用例开始，评估测试结果，如果发现实际结果与预期结果不一致，则这种不一致就是一个症状，它表明在软件中存在着隐藏的问题。调试过程试图找出产生症状的原因，以便改正错误。下面介绍常用的调试方法。

① 蛮力法。蛮力法的调试可能是为了找到错误原因而使用的最普通但是最低效的方法。当所有其他的方法都失败的情形下，才会使用这种方法。根据“让计算机自己来寻找错误”的思想，通过输出内存映象及程序中设置输出语句等手段，获取运行时现场信息。虽然大量的信息可能最终导致成功，但是更多的情况下，只是浪费精力和时间。

② 回溯法。回溯法是在小程序中经常能够奏效的相当常用的调试方法。从发现症状的地方开始，手工向回跟踪源代码，直到发现错误原因。不幸的是，随着源代码行数的增加，潜在的回溯路径可能会多到无法管理的地步。

③ 原因排除法。原因排除法是通过演绎和归纳以及二分法来实现的。对和错误发生有关的数据进行分析可寻找到潜在的原因。先假设一个可能的错误原因，然后利用数据来证明或者否定这个假设。也可以先列出所有可能的原因，然后进行检测来一个个地进行排除。

上面的每一种方法都可以使用调试工具来辅助完成。可以使用许多带调试功能的编译器、动态的调试辅助工具(“跟踪器”)、自动的测试用例生成器、内存映象工具以及交叉引用生成工具。

为了提高效率，在生成可执行程序时，应该重新编译改动过的源文件，而没改过的源文件就不必编译了，如在 Visual C++ 6.0 中可使用的预编译头文件就可达到上述要求。

(4) 连接生成可执行程序。

## 1.3 程序测试初步

软件质量与大多数工业产品一样需要通过检测手段来进行保证。软件测试通过对需求规格说明、软件设计说明和程序代码等进行复审，以保证在软件产品交付前，尽可能发现软件中潜在的问题。软件测试具有测试工作量大、难以进行“穷举”测试等特点。

测试(Test)是为了发现程序中的错误而执行程序的过程，所以好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案，成功的测试是发现了至今为止尚未发现的错误的测试。测试目的是想以最少的时间和人力，系统地找出软件中潜在的各种错误和缺陷。如果成功地实施了测试，就能够发现软件中的错误。测试的附带收获是：它能够证明软件的

功能和性能是否与需求说明相符合。测试过程中收集到的测试结果数据则是进行可靠性分析的重要依据。

测试对象除了程序代码外，还包括开发过程中产生的多种开发文档。这里主要讨论程序的测试。

考虑到软件本身是无形的逻辑实体的特性，最理想的是进行程序正确性的完全证明，遗憾的是除非是极小的程序，至今还没有实用的技术证明任一程序的正确性。为使程序有效运行，测试与调试是唯一手段。

### 1.3.1　程序测试的基本概念

应该在测试工作真正开始前的较长时间内就进行测试计划。一般而言，测试计划可以在需求分析完成后开始，详细的测试用例定义可以在设计模型被确定后立即开始，因此，所有测试可以在任何代码被编写前进行计划和设计。帕累托原则(Pareto Principle)告诉我们，测试发现的错误的 80%都集中在 20%的程序模块中。对发现错误较多的程序段，应进行更深入的测试。

测试应从“小规模”开始，逐步转向“大规模”，即从模块测试开始再进行系统测试。穷举测试是不可能的，因此，在测试中不可能覆盖路径的每一个组合，然而，充分覆盖程序逻辑，确保覆盖程序设计中使用的所有条件是有可能的。

下面介绍与软件测试相关的术语。

- 错误：对应英文为 Error。程序员在编写代码时会出错，这种错误称为 bug。随着开发过程的进行，存在的错误会不断地放大。
- 缺陷：对应英文为 Default。缺陷是错误的结果，更精确地说是错误的表现。
- 失效：对应英文为 Failure。在缺陷运行时，常常会发生失效的情况。一种是过错缺陷对应的失效；一种是遗漏缺陷对应的失效。
- 测试：对应英文为 Test。测试是一项采用测试用例执行软件的活动，在这项活动中某个系统或组成的部分将在特定的条件下运行，然后要观察并记录结果，以便对系统或组成部分进行评价。
- 测试用例：对应英文为 Test-Case。测试用例是为特定的目的而设计的一组测试输入、执行条件和预期结果。
- 回归测试：对应英文为 Regression Testing。回归测试的目的是测试由于修正缺陷而更新的应用程序，以确保彻底修正了上一个版本的缺陷，并且没有引入新的软件缺陷。

### 1.3.2　软件测试的技术分类

软件测试技术有多种分类，随角度不同而不同。

#### 1. 静态测试和动态测试

从是否需要执行被测软件的角度，可分为静态测试和动态测试。

(1) 静态测试

那些不利用计算机运行被测程序，而是通过其他手段达到测试目的的方法称作静态测试。人工测试就内容方面可对需求分析说明书、软件设计说明书、源程序做结构检查、流图分析等来找出软件错误。以下列举出最常用的静态测试方法。

- 桌面检查(Desk Checking)：由程序员自己检查自己编写的程序。桌面检查可视为由单人进行的代码检查，即由一个人阅读程序，对照错误列表检查程序，对程序推演测试数据。但其效率比较低，其主要原因在于：它是一个完全没有约束的过程，且人们一般不能有效地测试自己编写的程序。因此桌面检查最好由其他人而非该程序的编写人员来完成。总之，桌面检查胜过没有检查，但其效果远远逊色于其他方法。
- 代码审查(Code Inspections)：由程序员和测试人员组成一个评审小组，通过阅读、讨论对源程序代码进行静态分析的过程。首先由小组负责人提前将分析设计资料及程序文本等分发给其他成员，供研读后评审使用。在程序评审会由程序员仔细讲解程序逻辑，其他人员提出问题进行讨论分析，排查错误隐患。

(2) 动态测试

通过选择适当的测试用例，实际运行所测程序，比较实际运行结果和预期结果，以找出错误。下面将要介绍的黑盒测试和白盒测试就属于动态测试。

**2. 黑盒测试和白盒测试**

从软件测试用例设计方法的角度，可分为黑盒测试(Black-Box Testing)和白盒测试(White-Box Testing)。测试任何产品都可用这两种方法，即黑盒测试和白盒测试。这里仅引入基本概念，关于它们的测试用例设计方法在后面详细介绍。

(1) 黑盒测试

它只检查程序功能是否能按照规格说明书的规定正常使用，程序是否能适当地接收输入数据并产生正确的输出信息，程序运行过程中能否保持外部信息的完整性。黑盒测试又称为功能测试。黑盒测试方法主要有等价类划分、边值分析、因果图、错误推测等，主要用于软件确认测试。黑盒法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。

(2) 白盒测试

这种方法按照程序内部的逻辑测试程序，检测程序中的主要执行通路是否都能按预定要求正确工作。它是把程序看成装在一个透明的白盒子里，测试者完全知道程序的结构和处理算法。白盒测试又称为结构测试、玻璃盒测试。白盒测试的主要方法有逻辑覆盖、基本路径测试等，主要用于软件验证。

白盒测试法是穷举路径测试，全面了解程序内部逻辑结构、对所有路径进行测试。使用白盒测试方法时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。对于较大规模的程序，贯穿程序的独立路径数是天文数字，不可能进行穷举测试。

**3. 软件测试步骤**

按照软件测试的策略和过程分类，软件测试可分为以下四个步骤：

(1) 单元测试(Unit Testing)：单元测试又称模块测试、逻辑测试或结构测试。对软件中的基本组成单位进行的测试，发现各模块内部可能存在的各种差错。其测试对象是软件设计的基本单位——模块。

(2) 集成测试(Integration Testing)：集成测试也称组装测试、综合测试。是按设计要求把通过单元测试的各个模块组装在一起之后进行测试，以便发现与接口有关的各种错误。

(3) 系统测试(System Testing)：对已经集成测试后的软件作为计算机系统的一部分，与计算机硬件、支持软件、数据及人员元素结合起来，在实际运行环境下对软件进行彻底的测试，以发现潜在的软件错误。包括压力测试、容量测试、安全测试和性能测试等内容。

(4) 确认测试(Validation Testing)： 确认测试又称有效性测试、验收测试。确认测试主要由用户参加测试，检验软件规格说明的技术标准的符合程度，是保证软件质量的最后关键环节。

### 1.3.3 测试环境的构建

软件产品的测试设计与其本身结构设计一样具有挑战性，然而基于之前讨论过的一些原因，软件工程师经常将测试作为一种事后的措施，开发一些“感觉上正确”但缺乏完整保证的测试用例，对软件质量有不良影响。

目前，测试用例及其设计方法已经成为软件测试的重要概念和研究内容之一。测试用例被看作是有效发现软件缺陷的最小测试执行单元，也被视为软件的测试规格说明书，软件测试的本质就是针对要测试的内容确定一组测试用例。依据测试目标，要求设计出最可能发现最多数量的错误并耗费最少时间和最小代价的测试用例。

测试用例至少应该包括如下几个基本信息：

(1) 在执行测试用例之前，应满足的前提条件。

(2) 输入值。除了正常输入值外，关键是要寻找那些属于边界条件的输入值和不正常输入值。

(3) 预期输出(包括结果和实际输出)，是根据系统规格说明书来确定的输出结果、标准，有时是经过经验正确判断和理解所确定的。

表 1-1 给出了一个测试用例的需要信息的参考。

**表 1-1　测试用例记录卡**

| 测试用例 ID | | 测试用例名称 | |
|---|---|---|---|
| 目的 | | | |
| 前提 | | | |
| 输入数据 | | | |
| 预期输出 | | | |
| 结果 | | | |
| 执行历史 | | 执行人 | |
| 日期 | | 版本 | |

测试用例设计完成后，接着进行以模块为测试单元的具体测试。这就涉及测试环境的建立和模拟。

模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其他模块。先引入以下两个概念。

(1) 驱动模块(Driver)：调用测试单元的“主程序”，它接受测试数据，把这些数据传送给被测试的模块并打印有关结果。

(2) 桩模块(Stub)：是被测试模块单元所调用模块的代替模块，在模块调用接口、相关数据处理、控制返回等方面对被代替模块进行“模拟”。

在绝大多数应用中，一个驱动模块只是一个接收测试数据，并把数据传送给要测试模块然后打印相关结果的“主程序”。毫无错误的桩模块的功能是替代那些被本模块调用的模块。

驱动模块和桩模块都是额外的开销，这就是说，两种都属于必须开发但又不能和最终软件一起提交的软件，如果驱动模块和桩模块很简单，那么额外开销相对来说是很低的。不幸的是，许多模块使用“简单”的额外软件是不能进行足够的单元测试的。在这些情况下，完整的测试要推迟到集成测试步骤时再完成。

下面以大家熟悉的顺序查找算法作为测试对象，介绍测试环境的构建，具体见文件名为 searchseq.cpp 的程序，如图 1-4 所示。需要建立驱动模块进行测试，下面 test.cpp 的 main 函数承担驱动模块。在 main 函数中开始调用 search_seq 函数前，需要建立测试数据集及测试环境。

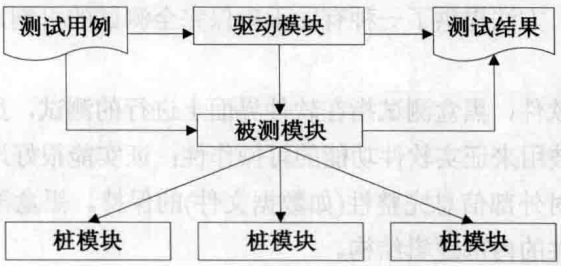



图 1-4 测试环境的构建

```
//文件名：searchseq.h
#ifndef _SEQSEARCH_H
    #define  _SEQSEARCH_H
    typedef int ElemType;          //假设数据元素为整型
    typedef int KeyType;           //假设查找关键字为整型
    typedef int SSTable[10] ;
    Search_Seq(SSTable ST, KeyType key);
    #define EQ(a,b) ((a)==(b))
#endif
//文件名：searchseq.cpp
    #include<stdio.h>
    #include"searchseq.h"
    Search_Seq(SSTable ST, KeyType key){     //顺序查找的算法，0 号元素为监视哨
        int i;
```

```
    ST[0]=key;
    for (i=sizeof(ST); !EQ(ST[i],key);--i);
    return i;
}
//文件名：test.cpp
#include<stdio.h>
#include"searchseq.h"
void main(){
SSTable st;
int i;
KeyType key;
int data[] = {15, 14, 11,13, 16,12};        //测试数据集
    for(i=0;i<sizeof(data);i++)
        st[i]=data[i];
    printf("请输入要查找元素");
    scanf("%d",&key);
    i=Search_Seq(st,key); //查找 13
    if (i)
        printf("\n 查找%d 元素位置：%d", key,i);
    else
        printf("\n 查找表中没有该元素");
}
```

### 1.3.4　测试用例设计

在过去的二十年，出现了大量的测试用例设计方法，为开发人员进行测试提供了系统的方法。更重要的是，方法提供了一种有助于确保完全测试的机制，并提供了揭示软件错误的最高可能性。

如果考虑计算机软件，黑盒测试指在软件界面上进行的测试，虽然设计黑盒测试是为了发现错误，它们却被用来证实软件功能的可操作性；证实能很好地接收输入，并正确地产生输出；以及证实对外部信息完整性(如数据文件)的保持。黑盒测试检验系统的一些基本特征，很少涉及软件的内部逻辑结构。

软件的白盒测试依赖对程序细节的严密检验，提供运用特定条件和/与循环集的测试用例，对软件的逻辑路径进行测试，在不同的点检验“程序的状态”以判定预期状态或待验证状态与真实状态是否相符。

一眼看去，可能认为全面的白盒测试将产生“百分之百正确的程序”，需要我们做的只是定义所有的逻辑路径、开发相应的测试用例，并评估结果，简而言之，详尽地生成用例以测试程序逻辑。然而，我们知道穷举测试缺乏可行性。

在实际使用白盒测试时，可选择有限数量的重要逻辑路径进行测试，检测重要数据结构的有效性。

测试用例使用((测试输入数据)，(预期结果))形式表示。

#### 1. 白盒测试技术

下面介绍几种用白盒方法设计测试数据的典型技术。