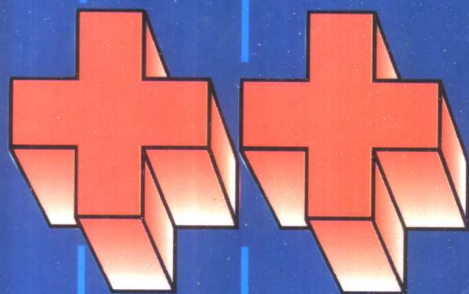


高等学校教材

张基温 编著

0
++

程序设计基础



高等教育出版社

高等学校教材

C++程序设计基础

张基温 编著

高等教育出版社

内 容 提 要

本书介绍了 C++ 的全集,该书从语法入手,同时强调程序设计的基本方法,以使读者能在较短的时间内,掌握 C++ 程序设计的基本技能和面向对象的概念和方法,并能编写出具有良好风格的程序。书中含有丰富的例题与习题,便于教学和自学。

本书内容丰富,结构紧凑,概念阐述清楚,注重学生能力培养。可作为高等学校有关专业程序设计语言课程的教材或参考书,也可供各类从事计算机应用的开发人员学习使用。

图书在版编目(CIP)数据

C++程序设计基础/张基温编著. —北京:高等教育出版社,1996 (2000 重印)
高等学校教材
ISBN 7-04-005655-0

I. C… I. 张… III. C 语言,C++—程序设计 IV. TP312C

中国版本图书馆 CIP 数据核字 (96) 第 05526 号

出版发行 高等教育出版社

社 址 北京市东城区沙滩后街 55 号

邮政编码 100009

电 话 010-64054588

传 真 010-64014048

网 址 <http://www.hep.edu.cn>

经 销 新华书店北京发行所

排 版 高等教育出版社照排中心

印 刷 北京市联华印刷厂

开 本 787×1092 1/16

版 次 1996 年 5 月第 1 版

印 张 22.75

印 次 2000 年 4 月第 6 次印刷

字 数 560 000

定 价 18.20 元

凡购买高等教育出版社图书,如有缺页、倒页、脱页等质量问题,请在所购图书销售部门联系调换。

版权所有 侵权必究

前 言

C++是一种多范型的程序设计语言,它既支持面向过程的程序设计,又支持面向对象的程序设计。它的成功在于务实与求新的结合,它没有抛开已经广受青睐的C从头重来,而是以C为支撑,向面向对象的领域延伸。这对已经熟悉C的人来说,如同一只手中已握着刀,而另一手又拿上了剑。对于面向对象程序设计思想的推广和普及,C++立了很大功劳。

有人说,“学习C++,必须先学好C”。这话虽有道理,但本人还是不敢苟同。固然C++是C的超集,学了C再学C++会很容易,但C++并不是简单地“在C上加了一点”,它是对C的改进和扩充,有些机制并不完全相同。为了学C++而先去学C,是不太经济的。因此,给那些想直接、快步进入C++的初学者提供一块垫脚石,正是本书的主要写作动机。

本书共六章,前三章介绍C++支持面向过程程序设计的基本机制,即相当于C语言的部分;后三章介绍用C++进行面向对象程序设计的基本方法。书中选用广泛流行的Borland/Turbo C++作为蓝本进行介绍。

尽管C++是一种适合组织较大型程序的程序设计语言,但对程序设计的初学者来说,首先应掌握的是一种程序设计语言的基本语法,其次是设计小程序的本领,不学会并熟悉设计小程序,就难以掌握大程序的设计方法。所以本书基本按语法结构编写,并注意对读者进行程序设计基本方法的训练。考虑本书的篇幅有限,为拓宽视野,进一步提高读者的程序设计能力并加深对重要语法的理解,本书还有一本姊妹篇——《C++程序设计基础例题与习题》。

国际标准化组织(ISO)C++标准化委员会技术专家蔡希尧教授、著名计算机教育家谭浩强教授分别审阅了本书。他们提出了宝贵的意见,并对本书的编写和出版给予了热情鼓舞和大力支持。在本书的编写和出版过程中还得到了西北大学卞雷教授的帮助。在此谨向他们深表衷心的感谢。

由于时间仓促,水平有限,错误与不妥之处难免,期待各位专家、同仁和读者不吝批评指教。

张基温

1995.12

目 录

引论	1	1.4.2 if...else 选择结构	38
0.1 程序=模型+表现	1	1.4.3 switch 选择结构	41
0.1.1 问题-模型-程序	1	1.4.4 while 重复结构	44
0.1.2 程序的正确性;模型的有效性 +表现的忠实性	2	1.4.5 do...while 重复结构	54
0.1.3 程序设计中的科学思维方法	2	1.4.6 for 重复结构	55
0.1.4 功能分析与结构分析	4	1.4.7 流程转向控制	58
0.1.5 模块:接口与实现	6	1.4.8 程序中止函数 exit()	61
0.1.6 数据结构与算法	7	1.5 程序测试及其用例设计	62
0.1.7 面向过程的程序设计(POP)与 面向对象的程序设计(OOP)	9	1.5.1 概述	62
0.2 C++	12	1.5.2 结构测试技术	62
0.2.1 C++概述	12	1.5.3 功能测试技术	65
0.2.2 C++程序开发的基本过程	13	习题	68
0.2.3 Borland/Turbo C++系统特点	13		
第一章 过程初步	15	第二章 过程的组织和管理	72
1.1 C++过程程序构成	15	2.1 用函数组织 C++过程	72
1.1.1 过程程序的功能	15	2.1.1 概述	72
1.1.2 C++过程程序结构	16	2.1.2 函数结构与函数定义	74
1.1.3 C++语素	17	2.1.3 函数名重载	76
1.2 常量与变量	19	2.1.4 内嵌函数	77
1.2.1 字面常量	19	2.1.5 函数原型与函数声明	77
1.2.2 程序变量	21	2.1.6 函数调用	78
1.2.3 符号常量	22	2.1.7 递归函数	80
1.3 运算符与表达式基础	23	2.1.8 C++库函数	85
1.3.1 算术运算符与算术表达式	24	2.2 程序实体及其存储类	87
1.3.2 关系运算符与关系表达式	25	2.2.1 程序实体的创建与生存期	87
1.3.3 逻辑运算符与逻辑表达式	26	2.2.2 作用域与可见性	88
1.3.4 条件运算符与条件表达式	28	2.2.3 C++存储类	89
1.3.5 位运算符及其表达式	29	2.3 编译预处理	98
1.3.6 赋值运算符与赋值表达式	30	2.3.1 宏定义	98
1.3.7 逗号运算符及其表达式	33	2.3.2 文件包含	100
1.3.8 关于表达式运算顺序的讨论	34	2.3.3 条件编译	100
1.4 语句及其流程控制	36	2.4 连接	102
1.4.1 语句	36	2.4.1 标识符的连接属性	102
		2.4.2 头文件	103
		2.4.3 C++模块与C模块连接	105
		2.5 工程文件	106

2.5.1 用工程管理器开发一个 工程的步骤	106	3.8.2 引用参数	178
2.5.2 工程管理器的使用技巧	107	3.8.3 返回引用的函数	178
习题	107	3.9 枚举类型	179
第三章 数据类型	112	3.9.1 枚举是用户定义类型	179
3.1 基本类型	112	3.9.2 枚举是一组被命名的 整型常量集合	180
3.1.1 整类型	112	3.9.3 枚举变量的运算与应用	181
3.1.2 字符类型	114	习题	182
3.1.3 实类型	118	第四章 类与对象	190
3.1.4 算术类型转换	120	4.1 类的定义	190
3.1.5 sizeof 运算符	121	4.1.1 类的组成与接口	190
3.2 数组类型	122	4.1.2 类的实现	192
3.2.1 一维数组	123	4.2 对象的创建与撤销	194
3.2.2 字符串	127	4.2.1 对象声明	194
3.2.3 多维数组	129	4.2.2 构造函数	194
3.3 指针类型	131	4.2.3 释放函数	197
3.3.1 地址与指针	131	4.2.4 对象创建时的内存动态分配	202
3.3.2 指针类型与指针声明	132	4.2.5 用对象初始化新对象:复制 构造函数	204
3.3.3 指针的运算	136	4.2.6 对象成员	209
3.3.4 动态内存分配	138	4.3 对象行为与性能控制	211
3.4 数组与指针	139	4.3.1 友元	211
3.4.1 数组名指针	139	4.3.2 共用体与共用体成员	216
3.4.2 数组元素的指针形式	143	4.3.3 静态成员	222
3.4.3 指向数组的指针变量	144	4.3.4 const 对象	226
3.4.4 字符型指针	147	4.4 对象组织	227
3.4.5 指针数组	149	4.4.1 对象向量	227
3.5 声明	152	4.4.2 指向对象的指针与对象链表	228
3.5.1 声明符	152	4.4.3 this 指针	232
3.5.2 复杂声明	153	4.5 对象运算与运算符重载	234
3.5.3 类型定义符:typedef	154	4.5.1 运算符重载的成员函数方式与 友元函数方式	234
3.6 函数与指针	155	4.5.2 类型转换与转换函数	237
3.6.1 指针参数	155	4.5.3 对象赋值与赋值运算符重载	239
3.6.2 带参主函数与命令行参数	159	4.5.4 下标运算符与函数调用 运算符重载	242
3.6.3 返回指针的函数	161	4.5.5 插入/提取符及其重载	249
3.6.4 指向函数的指针	163	4.5.6 运算符重载规则	253
3.6.5 指向 void 类型的指针	166	4.6 类模板	255
3.7 类属	168	4.6.1 类模板的定义	255
3.7.1 类型参数化	168	4.6.2 类模板中的友元函数	258
3.7.2 函数模板(template 函数)	169	4.6.3 异常处理	262
3.7.3 异常处理	173		
3.8 引用类型	174		
3.8.1 引用的声明与特点	174		

习题	263	6.2.2 基本提取操作	323
第五章 继承	266	6.3 插入/提取格式控制	325
5.1 派生类	266	6.3.1 流的格式状态字	325
5.1.1 public 派生与 private 派生	266	6.3.2 ios 类的格式化方法函数	325
5.1.2 Protected 成员与 protected 派生	269	6.3.3 预定义的 I/O 操纵算子	332
5.1.3 多基派生	270	6.3.4 自定义 I/O 操纵算子	333
5.1.4 派生类的构造函数与释放函数	271	6.4 流的出错处理	336
5.1.5 虚基类	276	6.4.1 流的出错状态	336
5.2 类层次中的访问规则	278	6.4.2 测试与设置出错状态位的函数	337
5.2.1 类层次中成员名的作用域	278	6.5 文件	338
5.2.2 类层次中的类转换	281	6.5.1 文件流	338
5.3 虚函数	286	6.5.2 文件的打开与关闭	338
5.3.1 方法的多态性与虚函数	286	6.5.3 文本文件的读/写	342
5.3.2 虚函数的访问	290	6.5.4 二进制文件的读/写	343
5.3.3 纯虚函数与抽象类	295	6.5.5 文件的随机读/写	345
5.3.4 虚释放函数	298	6.5.6 设备文件的使用	348
5.3.5 多基派生中虚函数的二义性	302	习题	348
5.4 容器类库	305	附录 Borland C++ 集成环境概要	349
5.4.1 Object 类库	305	1. IDE 功能	349
5.4.2 BIDS 模板库	309	2. 系统安装	350
习题	312	3. IDE 的启动与退出	350
第六章 I/O 流	319	4. IDE 窗口操作	351
6.1 概述	319	5. 编辑器命令	352
6.1.1 文件、缓冲区与流	319	6. 调试、执行命令	353
6.1.2 三种层次的 I/O 机制	320	参考文献	354
6.2 基本插入/提取操作	322		
6.2.1 基本插入操作	322		

引 论

0.1 程序 = 模型 + 表现

0.1.1 问题-模型-程序

程序设计涉及客观世界、认识世界、计算机世界。这三个世界的关系如图 0.1 所示。客观世界中的问题,在认识世界中是模型,在计算机世界中是程序。

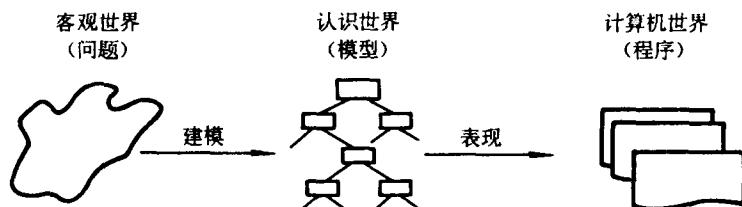


图 0.1

模型是对问题进行抽象的结果。它忽略了人们所不关心的方面或不必要的细节,保留了所关心的重要内容,以使人们深刻理解问题的本质,掌握解题的基本途径。如对“原来有三头牛,又买了两头,现共有几头牛”这样的问题,可以抽象为

$$3+2$$

这是一个忽略了具体物体本身、只研究其数量关系的模型。

程序是计算机理解并可执行的指令序列。当一个问题的模型,用某种计算机语言表现后,便得到了解决问题的程序。如上述模型用C++可表现为

```
#include <iostream.h>
main()
{
    int x = 3;
    x = x + 2;    // 或 x += 2;
    cout << "The sum of cow is:" << x << endl;
}
```

概括地说,程序设计是建立问题的模型并把它用一种计算机语言表现出来的过程。实际上,当问题的规模(复杂性)较大时,由于人的智力的不足,建模与表现的过程都是逐步向前推

进、并且常常是反复进行的,如图 0.2 所示。

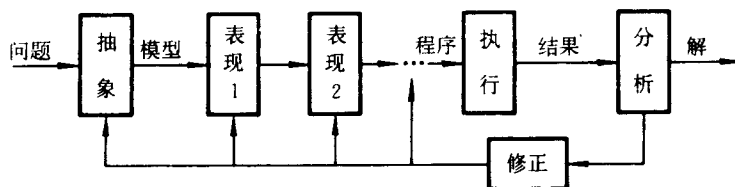


图 0.2

0.1.2 程序的正确性:模型的有效性+表现的忠实性

程序可以从许多方面评价,而正确性是最起码的。程序的正确性主要依赖于模型的有效性和表现的忠实性。

1. 模型的有效性

模型的有效性即模型表示实际系统的充分性。它主要用模型所产生的数据(结果)与实际系统所产生的数据(结果)的符合程度来评价。从强度上可以把模型的有效性分为如下三个层次:

- (1) 复制有效性。模型产生的数据与实际系统产生的数据相匹配。
- (2) 预测有效性。模型在从实际系统取得数据之前,能得到(至少“看出”)匹配数据。这是比复制有效性稍强的有效性。
- (3) 结构有效性。模型不仅能复制被研究的实际系统的行为,而且能真实地反映实际系统产生这个行为的机制。这是更强一级的有效性。

2. 表现的忠实性

表现的忠实性即程序实现模型的忠实性。

从方法学的角度,为了提高模型的有效性,人们除要熟悉并正确地应用问题领域内有关知识外,还应科学地运用智力,合理地分割问题,由粗到细,使每一步都在智力容易解决的范围内进行思考。为了提高表现的忠实性,除要正确地使用表现语言(包括中间语言)、认真地应用设计方法外,还希望缩短表现过程,减少转换层次。

程序的正确性与程序设计的效率是矛盾的。目前,人们已找到解决这一矛盾的有效途径,即提高软件的可重用性。具有可重用性的程序可以(或经简单修改)作为别的软件的构件。这样既可以缩短软件开发时间,又容易保证软件的可靠性。

0.1.3 程序设计中的科学思维方法

程序设计是涉及数学、计算机、特定问题领域、思维科学等的一门技术学科,是一种高强度的智力密集活动。它要求人们在这一过程中必须合理地组织智力,科学地进行思维。著名计算机科学家 Dijkstra 曾说过:“我有一个小小的脑袋,但是我要靠它去生活。”这是对程序设计哲理的精彩概括。下面介绍几种在程序设计中经常运用的科学思维方法。

1. 抽象

程序设计是一个由具体到抽象再到具体的过程。抽象强调问题内在的、本质的特性,而暂

不考虑其细节。这是人们认识、理解、处理复杂现象的最有力的思维武器。与复杂的现实世界相比,处于历史长河中一个局部的人们,其思维能力是有限的。由于思维能力的限制,人们对复杂性太高的问题,不可能一下子触及到其细节,做出精确思维。这时,有效的方法是先用高级的抽象概念构造、理解它,然后再用较低的概念构造、理解它的尚未被精确认识的部分;如此不断进行,形成不同的抽象层次,直到把问题精确构造、理解为止。例如,要做一张桌子,首先要决定桌子的大小、大致式样,然后再决定用什么材料、结构方式等。解题是一个由具体到抽象再到具体的过程。程序设计时,首先要将具体问题抽象为模型,再把模型表现为用具体的计算机语言表现的程序。在这一过程中,要求人们不首先考虑实现细节,而应将精力放在建立系统的总体模型上。这时,程序被看作由一个一个的程序零件构成。最后考虑各个程序零件的实现:或自己设计,或用已有的。

各种高级程序设计语言中,都引进了一些抽象机制,如数据类型、函数等,将服务与实现分离,以便能用一个名字指称具有相同性质的实现,使用户不须再关心实现的细节,能较大地缩短表现过程,有效地提高程序的正确性和程序设计的效率。

2. 归纳-演绎法

归纳是从个别发现一般概念,去异求同的思维方法。演绎是从一般的概念、原理认识个别的思维方法。

任何客观个体既有个别性,又有一般性,是个别性与一般性的统一体。归纳过程是从个别到特殊,从特殊到一般的思维运动;演绎是从一般到特殊,从特殊到个别的思维运动。它们是统一的人类认识过程中的相互对立又相互联系两种思维运动。在辩证的思维运动中,归纳以演绎为前导,演绎以归纳为基础。它们互为前题、互相促进。例如,张家的狗用于看门,李家的狗也用于看门……,可以归纳出“狗可用于看门”这一结论;又从有的狗可看门,有的狗可表演,有的狗可用于侦探……,归纳出“狗可以被训练”这一结论。由“狗可以被训练”这一特征可以演绎出:可以让狗去送信。

数学归纳法是一种典型的归纳思维方法。如,一般说来,对多个数求和要对各数依次相加。但是,对一个连续的自然数组成的数列中的各项依次相加,如计算

$$1+2+3+\cdots+10$$

时,可以按图 0.3 所示方法进行简捷计算

$$10/2 \cdot (10+1)$$

而要计算

$$1+2+3+\cdots+100$$

时,可以进行简捷的计算

$$100/2 \cdot (100+1)$$

于是,可以归纳出方法

$$\sum_{i=1}^n i = n/2 \cdot (n+1)$$

由这个一般方法,可以演绎出 n 为任何正整数时的计算方法来。

3. 分析-综合法

世界是复杂的,但是世界是构造的、可分的。分析-综合法就是基于世界可构造性与可分性的一种科学思维方法。分析是把整体分割成部分,把一个复杂系统分解成简单要素,把一个过

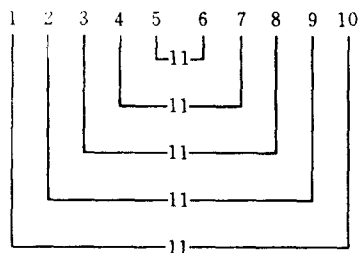


图 0.3

程划分成片断,或把一个动态过程离散成为一个静态序列来研究的一种思维方法。

综合方法是分析方法的发展,它是从认识世界的角度,把客观世界的各个部分、各个方面、各种因素结合起来,动态地进行考察、研究的一种方法。综合以分析为基础,只有经过对客观世界周密的分析,对其各个部分清楚地认识,才能正确地进行综合。但是,综合不是简单地将分析的结果还原,而是从理性角度,用某种概念,或在更高的层次上说明客观世界中事物的本质及其运动规律。

综合不同于归纳。综合是从同一命题的细节中提炼出完整的概念或理论,而归纳是从已知论及未知。

分析常常要用抽象方法进行。例如,计算机是一种智力机器,为发明计算机,首先要对人的解题活动进行分析,得到计算机应具备计算、记忆、自我控制、输入、输出等五大功能;再研究如何实现这些功能,最后组装,进行整体评价,找出薄弱环节,作进一步改进。

为开发大程序,人们已经找到并正在寻找更好的分析方法。关于这方面的知识请看有关资料。

4. 比较-分类法

分类是分析的一种主要方法或称特例。

1969年,比利时布鲁塞尔学派的领导人 I. Prigogine 在一次“理论物理学和生物学”国际会议上指出,世界总是不断变化和发展的,自然界应该存在着从无序向有序转化的客观规律。

有序与无序是相对的,它反映了人们认识客观世界的绝对性与相对性。世界是物质的,物质是运动的,运动是有规律的,因而世界是可以被认识的。这就是世界有序的绝对性。人们对客观世界的认识是螺旋式向前推进的。这就是有序的相对性。

世界的有序性,还表现在组成它的成分是有类型的。1984年德国数学家 P. Martin-Löf 提出了一个类型论(Type Theory)。他认为

- 世界是按类型构造的
- 类型是分层次的
- 低层次类型是较高层次类型的一个成员,可以表示成

$$U_{i-1} \in U_i$$

类型论是人们对客观世界有序性认识的一个侧面。现实世界中各种事物、现象、概念的差异性和同一性(相似论)是类型学的客观基础。而抽象概念是分类的主观依据。通过比较可以突出客观事物、现象间的差异;通过分类,可以使客观事物在人们主观的认识概念上条理化、系统化、整体化。

一个系统中的元素,按不同的抽象概念,可以进行不同的分类;可以用抽象的方法自顶向下地进行,也可以用归纳的方法自底向上地进行。如领导分配甲去调查运输工具的市场情况,分配乙去调查纺织品的市场情况,这时的分类是自顶向下的;而甲调查运输工具的市场情况后,可以按机动与人力,也可以按陆运、水运、空运,或按所属部门对调查结果进行分类,这时的分类是自底向上的。

0.1.4 功能分析与结构分析

计算机程序的执行,实际上是在计算机上对问题所描述的现实世界中运动的模拟。从哲学的观点看,模拟可以按功能模拟与结构模拟两个方面分别或交叉进行。模拟是在对被模拟事

物进行相应分析的基础上进行的。

1. 功能分析与功能模型

功能模拟只强调系统应具备哪些功能,而不强调这些功能必须用什么样的构造形式实现。如当前的计算机就是从功能模拟的角度,被认为是一种模拟人脑运算的机器,而不是从人脑神经的角度来实现信息处理机能。再如,洗衣机的发明也是基于功能模拟的。

用抽象思维进行功能分析的基本方法是,首先应把系统看成一个黑箱,抽象出它的输入与输出,中间的部分是看不见的,只表示一种功能(或处理),如图 0.4 所示。对计算机来说,输入的是原始数据,输出的是结果。对洗衣机来说,输入的是脏衣物,输出的是净衣物。

把一个黑箱表达清楚以后,便可以打开它,看它是否可以进一步分解成几个更基本的功能……。如此不断把抽象级别降低,便可以由上而下地得到一个系统的功能层次图。图 0.5 为计算机功能的层次图。功能层次结构图最低一层应当是一些单一的、极易实现的功能。

2. 结构分析与对象模型

结构分析主要研究系统组成,它强调模型的结构与客观世界的结构相对应,认为世界是由各种各样具有一定运动规律和内部状态的对象所组成的:对象(object)是现实世界中的有意义的事物,可以是物质的,也可以是一种概念;复杂对象由简单对象组成;每个对象有自己的属性与行为,不同对象间的相互作用和通信构成了现实世界。因此结构分析围绕着现实世界中的对象来构造系统模型,而不是围绕功能来构造系统模型。或者说,它强调首先要了解系统由什么组成,而不是象功能分析那样强调首先了解系统要干什么。

分类是对复杂系统进行结构分析的主要方法。通过分类对组成系统的对象进行抽象,可以简化系统的模型,并容易理解系统的实质。属于同一类型的对象有类似的属性、共同的行为(操作)、与别的对象有类似的关系、共同的语义。例如,不管在哪个坐标点上,划出一个什么样(半径,颜色)的圆,都有相似的属性(半径,圆心坐标,颜色)、共同的行为(被平移,被放大,被显示,被选择等)。具有一组类似性质的对象,称为对象类。

类具有层次关系。在下页图 0.6 所示“图形(Figure)”类层次结构图中,每个对象类由类名、属性、方法(行为)三部分来描述。在类层次结构中,父类是子类的超集,即父类所描述的对象包含了其任一子类所描述的对象。或者说,子类比父类具有更多的属性,以表明子类对象的特殊性。如一个圆类比一般二维图的共同属性要多一个属性(直径 diameter)。也可以说,父类派生了子类,或子类继承了父类的性质。这种模型也称对象模型。

在对系统进行结构分析时,为了深刻理解对象行为,还要对对象进行动态分析,建立对象的动态模型,作为对象模型的补充。

系统分析与问题定义是建立模型的基础。从模型的有效性看,功能模型最高可以达到预测有效性,而结构(对象)模型可以达到结构有效性。

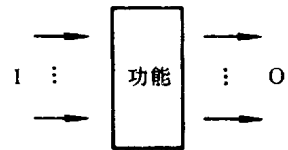


图 0.4

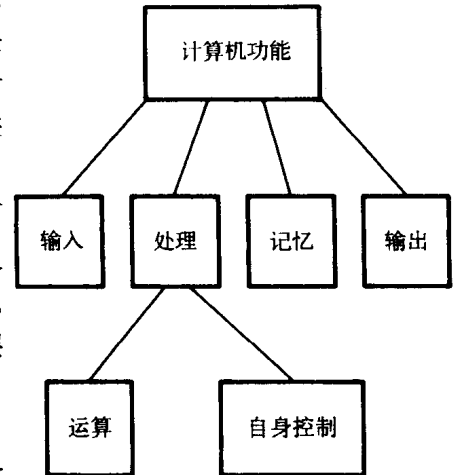


图 0.5

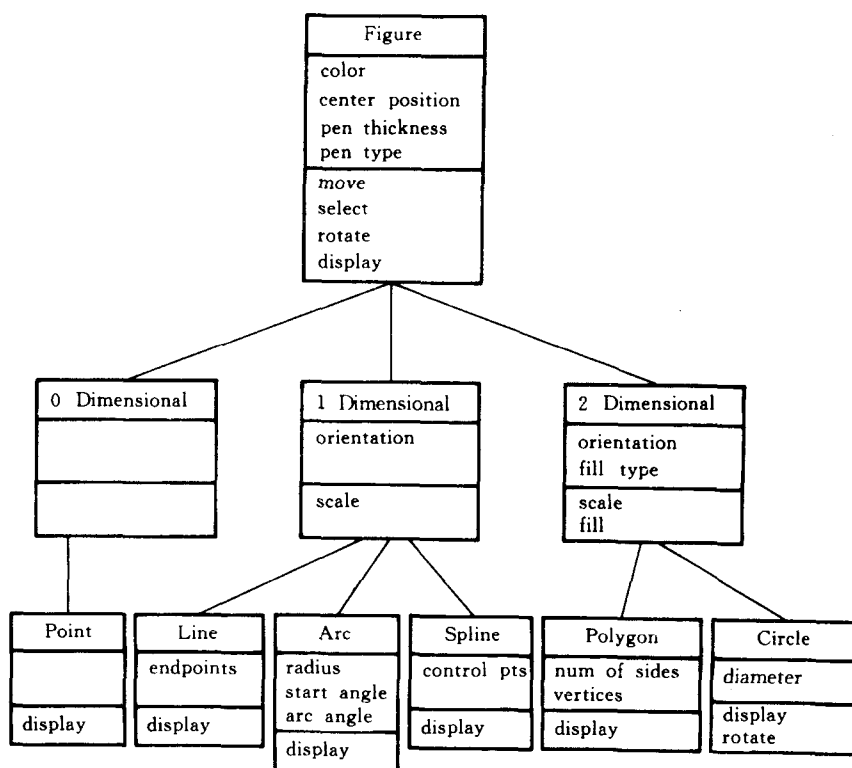


图 0.6

0.1.5 模块：接口与实现

分析(功能分析或结构分析)得到系统模型的总体结构。在功能或类层次结构图中,基本元素称为模块(在功能层次图中为功能模块,在类层次图中为类模块)。每个模块与其它模块的联系称为接口。在程序设计中,模块之间的联系太多,会造成模块修改与维护的困难,也会造成错误在模块间的蔓延。为了能得到一组好的模块,应当使每个模块中的信息,对不需要这些信息的其它模块是不应该访问的。这称为模块设计的信息隐蔽原理。

在功能层次结构图中,每个模块的接口需要指明以下几点:

(1) 调用关系,只有父模块可以调用其子模块。

(2) 数据传送关系:调用模块(父模块)向被调模块(子模块)传送数据(参数),作为被调模块的输入;被调模块执行后,向调用模块返回一个值,作为被调模块的输出。在类层次图中,各模块是类。按照信息隐蔽原理,每个类对象中的信息,(数据及方法,统称类的成员)应分为公开的(public)和私有的(private)两部分。

因此,定义一个类的接口需要说明以下几点主要内容:

(1) 该类有什么继承关系,如图 0.6 中的类 2 Dimensional 继承了类 Figure 的特性;

(2) 该类除继承的特性外,自己还有哪些特殊性;

(3) 该类的成员中,哪些成员是外部可以访问的,哪些是外部不可访问的。

接口定义好后,接下来的工作是实现模块。所谓实现,通常含有两层意思:构造模块的内部细节与将模块用某种计算机程序设计语言表现。对功能模块来说,每个模块描述了该功能实现的过程;对类模块来说,每个主要模块由数据成员(描述属性)与方法成员(描述行为)两部分组成。

应当特别指出的是,如果有现成的、可重用的模块,就不必自己重新来构造模块了。

0.1.6 数据结构与算法

软件模块要用数据结构与算法实现。

1. 数据结构

数据是客观事物属性的抽象。为了有效地对数据进行处理,不仅要研究每个数据取什么值,而且要研究如何组织数据。数据结构是现实世界中事物属性及相互联系的模型。对于简单的问题,数据量较少且相互间的关系比较简单,程序中采用简单变量;当数据量较多或相互间的关系比较复杂时,采用较复杂的数据结构,如集合、向量与数组、堆栈与队列、树、图等。

2. 算法

一个算法是由一些特定的操作按一定的规则组成的有穷序列。它由可理解与可执行的基本操作集合与对操作顺序进行控制的两大要素组成。

(1) 基本操作

不同的问题空间中有不同的运动形式。或者说对不同的数据所施加的操作种类不同。但是用计算机解题时,都要用计算机所能接受的操作来模拟。这些操作主要是

- 算术运算
- 逻辑运算
- 关系运算,即比较
- 赋值运算,向一个变量写入值

(2) 控制结构

控制结构可以使操作的执行顺序与书写顺序不一致,如对于 9,当下面的各操作

OP1, OP2, OP3, OP4

分别为/3, +6, $\times 2$, -1 时,图 0.7 中(a)与(b)的执行结果分别为 17 与 7。

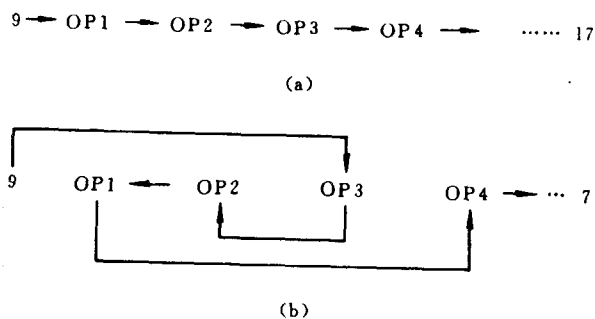


图 0.7

下面的三种控制结构是最基本的,称为算法的基本控制结构:

- 顺序结构:操作的执行顺序与书写顺序一致
- 选择结构:根据条件从两种以上的操作中选择一种
- 重复结构:在一定的条件下,重复执行某些操作

这三种基本结构可以形象地用图 0.8 所示的图形描述。这种图形称为流程图。流程图是一种算法描述工具,其中 S 表示操作,P 为判断条件。

1966 年 Bohm 和 Jacopini 证明,用三种基本结构可以构成任何复杂的算法。

例 0.1.2 将 A,B 两磁带内容交换的算法,如图 0.9 所示。temp 为一盘空带。

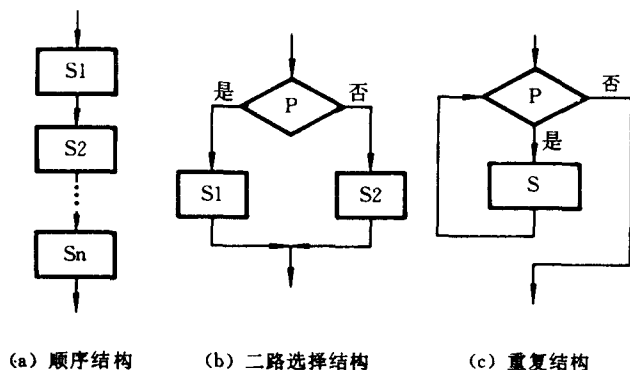


图 0.8

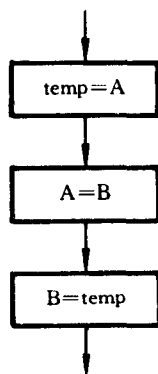


图 0.9

例 0.1.3 在三个数中求最大数。算法如图 0.10 所示,其中 max 代表最大数。

例 0.1.4 求两个正整数 m 与 n 的最大公约数,设 $m > n$ 。算法如图 0.11 所示。

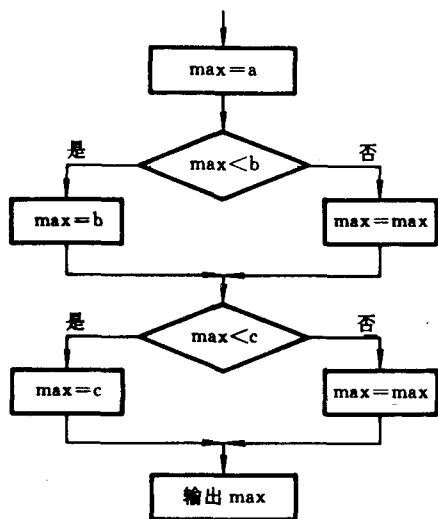


图 0.10

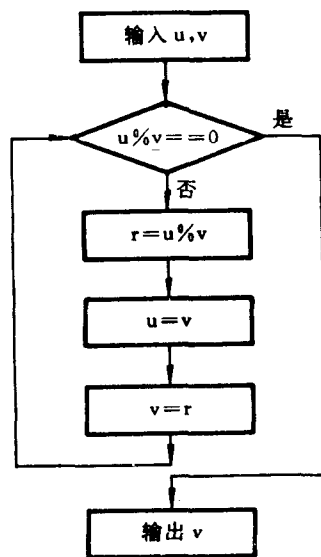


图 0.11

这是几个在简单的数据上施加算法的例子。随着问题规模的加大,数据结构及其上的算法的复杂程度也会加大。一般说来,合理的数据结构会降低算法的复杂性,而数据结构的实现又要以一定的算法为基础。还须说明的一点是,在构成程序的两大要素——数据结构与算法中,数据结构相对稳定,而对同一问题算法往往不是唯一的。

算法是与演绎相平行的一大数学方法。人们已经用公理建立了一个庞大的演绎数学系统(尽管在许多细节是用算法作补充的),随着计算机的普及,人们也正努力用算法代替它。按算法方法求解问题,是程序设计初学者应努力训练的重要方面。

0.1.7 面向过程的程序设计(POP)与面向对象的程序设计(OOP)

1. POP(Procedure-Oriented Programming)

对系统进行功能分析并得到一些按层次给出的“做什么”,而“如何做”要用一步一步的操作来实现。这些为了得到问题的解而执行的一步一步的操作,就称为过程。因此,过程的设计是基于算法的,每个过程的算法,就是这个过程的模型。

在程序设计语言中,过程常常被抽象为函数(如 C 语言)、子程序(如 BASIC 语言)或程序过程(如 Pascal 语言)。每个过程表示了一种加工机,能对不同的数据(参数)实行同样的操作。图 0.12(a)定义了一个名为“*”的过程,其功能是对输入的整数求积:当输入 5 与 3 时,输出 15。下面看另外一个例子。

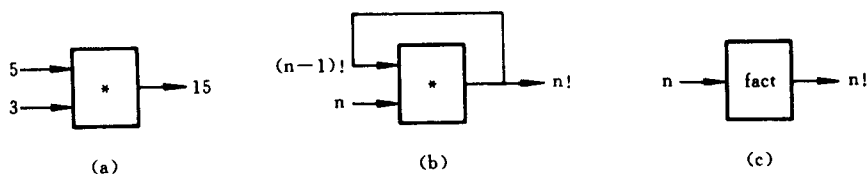


图 0.12

定义了功能后,再进一步考虑它们的实现。

例 0.1.5 求阶乘的过程。

解 按定义

$$n! = 1 * 2 * 3 * \cdots * n$$

如果,现在只有一个名为“*”的乘法器,则可以用下面的形式实现:

$$n! = n * (n-1)!$$

这是求阶乘 $n!$ 的一个模型。它给出了由 $(n-1)!$ 递推 $n!$ 的方法,如图 0.12(b)所示。用这个模型,依次输入 $1, 2, \dots, n$,便可以求出 $n!$ 。而“依次输入 $1, 2, \dots, n$ ”可以用一个简单的重复算法实现。再增加这部分机制,便可以抽象到图 0.12(c)所示的 fact。

面向过程的程序设计是一种基于功能分析的、以算法为中心的程序设计方法。这是一种传统的程序设计方法。它的开发过程一般包括如下步骤:

- (1) 系统分析:定义问题及用户需求;
- (2) 系统(初步)设计:设计问题的功能模型,定义各模型的界面;
- (3) 算法(详细)设计:设计各功能模块的算法模型;

- (4) 程序(编码)设计:将各模型用算法语言表现;
- (5) 程序测试;
- (6) 程序维护。

传统的程序设计语言是面向过程的,常用的主要有以下几种:FORTRAN,COBOL,Pascal,BASIC,Ada,C等。一般说来,一个面向过程的设计语言应具备如下机制:

- (1) 数据描述:变量、常量及其类型;
- (2) 完备的运算符与表达式规则;
- (3) 流程控制与模块化功能。

在面向对象的程序设计出现之前,对于一些非过程性的问题,必须按机器工作的步骤,设计过程化的模型。这样不自然的表现方式,不仅从根本上难以保证模型的有效性,同时增加了程序的复杂性。由于面向过程的程序设计是基于功能分析并以算法为中心,因此功能的多变性和算法的不唯一性,增加了程序的维护和测试工作量。统计表明,基于功能分析的软件费用中,维护费用一般要占到整个软件费用的70%左右,而在软件维护费用中,65.4%是由于环境和用户需求变化而引起的。原因就在于“功能”是以算法为中心的,用户对于一个系统的功能要求是极易变化的。并且,随着问题规模的增大,程序变得越来越难以管理。

60年代末期,为应付当时日趋严重的“软件危机”,人们提出了结构化程序设计方法。结构化程序设计一定程度地缓解了“软件危机”但并没有彻底解决。70年代末80年代初,整个软件界又蒙上一片愁云之时,面向对象的程序设计给人们带来新的希望。于是整个80年代面向对象空前活跃,许多学者、研究机构、软件开发商都开始向面向对象这面旗帜下靠拢。

2. OOP(Object-Oriented Programming)

相对于POP而言,面向对象的程序设计是一种基于结构分析的、以数据为中心的程序设计方法。在面向对象的程序中,活动的基本单位是对象,向对象发送消息可以激活对象的行为。所以有人也把面向对象的程序描述为

对象+消息传递

图0.13是一个整数对象接收消息时响应的示意图,图中:

- (a) 一个整数类创建一个整数对象5,这个对象具有+,-,*,factorial(阶乘)等行为;
- (b) 对象5接受发来的消息+3,用相应的行为响应,产生一个值8;
- (c) 对象5接受发来的消息factorial,产生一个值120。

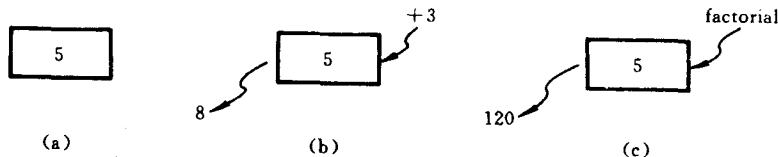


图 0.13

对照图0.12中所示过程模块可以看出:过程模块只能接收数据,如果把过程模块比作机器,则面向对象的程序模块可以比作人;“他”接收的是消息,并且能对不同的消息产生不同的行为。好象一个厨师接收“炒菜”的命令后能拿出菜、接收“做面”的命令能拿出面一样。