目　　录

第一章　软件设计概论

[内容提要] 本章简要讨论了计算机系统与软件设计，概述了软件工程的基本思想，介绍了现代程序设计的标准、程序设计的风格和现代程序设计方法。

1.1　计算机系统与软件设计

1. 计算机系统

一个计算机系统通常包含两大部分：硬件和软件。硬件是指系统中的实际设备，软件则包括运行机器所需的各种程序和相关的文件资料。

任何一个给定的计算机应用系统，至少应该包括三个部分：硬件、软件和数据。在程序的控制下，计算机把数据处理成信息。

系统是为了实现一个目标而共同工作的一组部件。例如，一个公司的工资发放系统，它的目标是为公司雇员支付工资。雇员们每天刷卡，把他们工作的时数记录在计算机上。在计算中心，雇员们的工作时数被读入工资程序，存为数据文件。当工资程序执行时，程序访问数据文件。最后，系统打印出各个雇员的工资单。为了使工资发放系统能够正常工作，人、磁卡机、数据文件、计算机硬件和软件都必须仔细地进行协调。仅仅编写程序是不够的，因为程序只是系统的一个组成部分。

不论是那种计算机，它只能识别和执行一些基本的命令，即指令。在计算机中，所有的指令和数据都是用二进制表示的。程序人员直接处理这些指令与数据是很麻烦的，而且极易出错。解决这一问题的途径是发展软件。

在计算机系统中，硬件是物质基础，软件是指挥枢纽，硬件通过软件发挥作用。

2. 软件

随着计算机技术的不断发展，软件的开发与生产已成为一种被称为软件工程的新兴产业。按照国际标准化组织(ISO)的定义，软件是指“与计算机系统的操作有关的计算机程序、过程、规则及任何与之有关的文档说明”，即软件是由计算机程序与相应的各种文档组成的。例如：研制计划书、设计说明书、使用说明书等都是程序文档的典型例子。程序文档在提高程序的可读性、可操作性、可维护性以及便于程序的交流等方面发挥着重要的作用。

软件一词通常有以下三层含义：

(1) 指程序和相关的文件与数据；

(2) 指特定的计算机系统中所有上述含义下的软件的整体；

(3) 指为了研究、开发以及维护前述含义下的软件所涉及的理论、原则以及技术所构

成的学科，即软件学，简称软件。

在计算机系统中，软件有以下三个作用：

(1) 作为用户与硬件的界面，使得用户可以通过软件与计算机进行交流。

(2) 作为计算机系统的指挥机构，控制系统的运行。

(3) 为计算机系统体系设计提供依据。

软件的发展经历了以下三个阶段：

(1) 1945－1955 年为第一阶段。这时的工作方式为串行工 作方式。用户使用低级语言编写程序。

(2) 1956－1967 年为第二阶段。这时的工作方式为多道并 行方式。用户使用高级语言编写程序。

(3) 1968 年至今为第三阶段。这时用户使用软件工程进行程序 设计。

随着硬件性能价格比和质量的不断提高，软件已经成为限制计算机系统发展的关键因素。

在计算机系统发展的早期所形成的一些错误概念和方法，已经严重限制了计算机软件的开发。特别是用错误方法开发出来的许多大型软件几乎根本无法维护，只好提前报废，造成了极大的浪费，并由此产生了所谓的“软件危机”。

3. 软件危机

软件危机是指在计算机软件的开发与维护过程中所遇到的一系列重要问题。它通常包含以下两个方面的问题：

(1) 如何开发软件，怎样满足对软件的日益增长的需求。

(2) 如何维护软件，怎样保证数量不断膨胀的软件的正常运行。

具体而言，软件危机的主要表现有：

(1) 对软件开发成本和进度的估计不准确。实际成本比估计成本有可能高出一个数量级，实际进度比预期进度拖延许久的现象屡见不鲜。

(2) 用户对开发出的软件不满意。这主要是由于软件开发人员对用户的要求只有模糊的认识，与用户之间的信息交流不够充分。

(3) 软件产品的质量不可靠。软件质量保证技术没有坚持不懈地应用到软件开发的全过程中。

(4) 软件难以维护。可重复使用的软件还是一个没有完全做到的、正在努力追求的目标。

(5) 软件的文档资料不全，使得软件的开发与维护出现许多问题和困难。

(6) 软件成本在计算机系统总成本中所占比例不断上升。

由于相当多的软件专业人员对软件开发和维护还有一些错误观念，例如在分析方法、设计方法方面等。在软件设计中不自觉地使用了错误的方法与技术，可能使软件问题发展成软件危机。

4. 系统分析与设计

因为人们需要信息，所以要开发基于计算机的系统。通常，用户知道需要什么，但是他们可能缺乏得到这些东西的计算机专门知识。而计算机专业人员，有这方面的知识，却

可能在用户的专业领域内缺少训练。对于用户要求没有完整准确的认识就匆忙着手编写程序，是许多软件开发工程失败的主要原因之一。

只有用户才真正了解他们自己的需要，但是，许多用户在开始时，并不能全面准确地叙述他们的需求。因此，软件开发人员必须做大量的深入细致的工作，反复与用户交流信息，才能真正全面、准确、具体地了解用户的需求。对问题和目标的正确认识是解决任何问题的前提和出发点，软件开发工程同样也不例外。

系统分析员是一种能把用户的需求转换成计算机技术术语的专业人员。因此，系统分析员是用户与计算机专业人员之间的桥梁。

系统分析员通常遵循一个意义明确、有条理的过程。该过程至少应该包括以下步骤：

(1) 问题定义；

(2) 分析；

(3) 设计；

(4) 实现；

(5) 维护。

系统是由系统分析员根据定义好的有组织的处理过程来进行规划和设计的。第一步是问题定义，分析员在这一步要准确地发现用户的需求。在初步的问题定义之后，通常要接着进行系统的可行性研究，以确定该问题是不是能够解决。

有了明确的问题定义和可行性分析后，就可以开始进行系统的分析工作了。在这个阶段，分析员要研究出系统的一个逻辑模型。当逻辑系统被用户审查通过以后，就可以开始系统的设计工作了。

在设计阶段，分析员要研究出物理系统的一个模型。随后进入实现阶段。在实现阶段要做的工作有：程序的规划与编写、硬件的订购与安装、以及最后的装配与调试等。当系统交付使用以后，维护工作就开始了。

1.2 软件工程概要

1. 软件工程

软件工程是指导计算机软件开发与维护的工程学科。它采用工程的概念、原理、技术和方法来开发与维护软件。

软件工程强调使用生命周期方法学和结构分析与设计技术。生命周期方法学是从时间角度，对软件开发和维护的复杂问题进行分解，把软件生命的漫长周期，依次划分为几个阶段。每个阶段有相对独立的任务。然后，逐步完成每个阶段的任务。

使用生命周期方法学进行软件开发时，一个阶段一个阶段地进行开发。前一个阶段任务的完成，是后一个阶段工作的前提和基础；后一个阶段任务的完成又是使前一个阶段提出的解决方法进一步具体化。每一个阶段的开始与结束都有严格的标准。对于任何两个相邻的阶段来说，前一个阶段的结束标准就是后一个阶段的开始标准。在每一个阶段结束之前都必须进行严格的技术审查和管理复审。

生命周期方法学把软件生命划分为几个阶段，每个阶段的任务比较简单，便于不同的

人员分工协作，从而降低了整个软件开发工程的难度。在软件生命周期的每个阶段，都采取科学的管理方法和良好的开发技术，并且坚持严格的审查制度。这使得整个软件开发工程以一种有条不紊的方式进行着，保证了软件的质量，也提高了软件的可维护性。因此，可以大大地提高软件开发的效率。

一般说来，软件生命周期分为：软件定义、软件开发与软件维护三个时期，每个时期又有几个阶段。

(1) 软件定义：这一时期的主要任务是，确定软件开发工程必须完成的总目标；确定工程的可行性；研究实现工程目标应该采用的方法；估计完成工程需要的资源和成本；制定工程进度表。软件定义时期通常进一步划分为问题定义、可行性分析和需求分析三个阶段。

(2) 软件开发：这一时期的主要任务是，具体设计和实现在问题定义时期定义的软件。软件开发时期通常进一步划分为总体设计、详细设计、编码与测试四个阶段。

(3) 软件维护：这一时期的主要任务是维护软件的正常使用。

2. 软件生命周期

以下概要介绍软件生命周期每个阶段的基本任务和结束标准。

(1) 问题定义：这一阶段的基本任务是确定要解决的问题是什么。通过本阶段的工作，应该得到关于问题性质、工程目标和规模的书面报告。由于只有用户才真正了解他们自己的需求，而许多用户在开始时并不能准确、具体地描述他们的需要。因此，系统分析员必须与用户进行足够的沟通，直至得出一份双方都满意的文档。本阶段的结束标准为：提交关于系统规模与目标的报告书。

(2) 可行性分析：这一阶段的基本任务是找到解决上一个阶段提出的问题的可行的方法，导出系统的高层逻辑模型，并对建议的系统进行仔细的成本效益分析。本阶段以给出系统的高层逻辑模型和成本效益分析为结束标准。

(3) 需求分析：这一阶段的基本任务是确定目标系统必须具备的功能。系统分析员在与用户密切配合、充分交流信息后，得出用户确认的系统逻辑模型。本阶段以给出系统的逻辑模型和算法描述为结束标准。

(4) 总体设计：这一阶段的基本任务是考虑如何解决系统分析时期提出的问题。通常至少应该考虑以下几种可能的方案：低成本的解决方案、中等成本的解决方案和高成本的完美方案。本阶段要推荐一个最佳方案，并且制订实现该方案的详细计划。本阶段以给出系统的可能解法和推荐的系统结构为结束标准。

(5) 详细设计：这一阶段的基本任务是，把总体设计阶段确定的比较抽象的解决问题的方法具体化，设计出程序的详细规格说明。本阶段以给出系统的编码规格说明为结束标准。

(6) 编码测试：这一阶段的关键任务是写出正确的、容易理解的、容易维护的程序模块。程序员应该根据目标系统的性质和实际环境，选取一种适当的高级程序设计语言，把前一个阶段设计的结果写成具体的语言编码，并且仔细测试编写出的每一个模块。本阶段以写出源程序清单和单元测试方案与结果为结束标准。

(7) 综合测试：这一阶段的关键任务是通过各种类型的测试与调试，使软件达到预定

的要求。应该用正式的文档资料把测试计划、详细的测试方案和测试结果保存下来，作为软件配置的一个组成部分。本阶段以给出综合测试方案与结果以及完整的软件配置为结束标准。

(8) 软件维护：这一阶段的关键任务是，通过各种必要的维护活动使系统持久地满足用户的要求。通常包括改正性维护、适应性维护、完善性维护和预防性维护四类维护活动。每一项维护活动都应该经过提出维护要求、分析维护要求、提出维护方案、审批维护方案、确定维护计划、修改软件设计、修改程序、测试程序和复查验收等一系列步骤。本阶段实际上是一次压缩和简化了的软件定义和软件开发的过程。每一项维护活动都必须准确地记录下来，作为正式的文档资料加以保存。本阶段以给出完整准确的软件维护记录为结束标准。

1.3 现代程序的标准

1. 程序设计质量

随着计算机制造技术水平的飞速提高，计算机内存量与运行速度大大提高了，人们不必为了节约很少的存储空间和运行时间，挖空心思地去想出一些人为的技巧，牺牲程序的可读性。在现代应用中，一个良好的程序应该具有良好的结构，逻辑清楚，易读，易懂，易操作。由于计算机硬件的价格正在不断下降，而随着应用水平的提高，软件的开发量和维护费用却在不断增长。如果程序设计得难以看懂，修改时会浪费很多时间，同时也不便于交流。从整体上看，这种浪费更大。因此，国际标准化组织(ISO)软件委员会提出了以下七条判别程序优劣的质量标准：

(1) 功能应满足要求；
(2) 可靠性应达到预定水平；
(3) 用户界面应友好；
(4) 在规定的条件下执行时间要短；
(5) 资源消耗要少；
(6) 可维护性要好；
(7) 可移植性要高。

其中正确性、易读性与高效性是最常用的标准。为了保证三者的有效统一，应该根据具体情况综合考虑各种因素，权衡利弊，确定主要目标，选择效果较好者。例如，从甲地到乙地共有三条路线相通，其中第一条路线最近，第二条路线次之，第三条路线最远。是否走最近的一条路线就是最经济、效益最高？并不一定。因为最近的路可能要翻山越岭，或者路况较差，而最远的路可能是高速公路，还有直达车辆。因此，应该根据道路好坏以及交通工具如何等综合条件来做决定。

评价一个程序，通常需要考虑以下四个方面的优劣：

(1) 正确性；
(2) 可读性、可移植性、通用性；
(3) 透明度；

(4) 效率。

正确性是不言而喻的。

可读性、可移植性、通用性与算法、程序结构、数据结构、语句结构等有关。通常结构良好的程序易读、易移植,并且通用性较强。

透明度是指在使用该软件时,用户需要对该软件的内部结构以及所涉及的数学物理知识所了解的程度。需要知道的越少,透明度越高。用户对软件本身的构造并不感兴趣,他们最关心的是软件的功能、效率和使用方法。功能越强、效率越高、使用越方便的软件,越受用户的欢迎。

效率通常指运算量与内存开销。对于解决同一个问题,所需运算量和内存开销越少,效率越高。

2. 文件规范

软件是由程序和相关文档组成的。为了便于程序的管理,加快程序的形成,便于程序的使用与维护,针对求解问题的流程,应该在每一个阶段形成一个规范的文件。通常,在程序设计的过程中,应该形成以下五个文档:

(1) 问题要求:包括待解问题的各项要求、指标,应当十分明确和详细的书写。这既是以后设计程序的依据,也是检验程序的标准。

(2) 算法分析:包括待解问题的数学模型、计算方法,指出求解该问题要使用的数据,这些数据的要求形式和处理方法。

(3) 程序设计:要使用结构化程序设计方法,自顶向下、逐步细化、模块化地设计程序。程序应有注解和详细的使用说明。

(4) 调试报告:准确记录在程序调试过程中所出现的问题和解决的措施,以及如何验证程序等。

(5) 总结报告:分析所用算法的特点,介绍程序的适用范围以及如何扩充程序的功能等。

这五个文档应该在整个工作过程中逐步形成,而不应该在程序全部调试完成后再形成。

除了在求解问题的过程中逐步形成文档外,在程序交付使用时,一般应有两个不同的文件:用户文件和技术文件。

用户文件通常包含以下内容:

(1) 程序名称;

(2) 对程序简短的描述;

(3) 程序需要输入的数据,包括这些数据的范围、格式和输入设备;

(4) 在有效数据下,程序的正常输出;

(5) 异常报告;

(6) 程序的限制;

(7) 在特定计算机上运行时所必须的命令序列;

(8) 程序编写者的情况。

技术文件通常包含以下内容:

(1) 程序名称及其作用;
(2) 程序设计的过程与结果,包括程序编写者的姓名和地址等;
(3) 程序总的结构;
(4) 每个模块的说明;
(5) 关键性数据结构的说明;
(6) 调试与测试方法。

1.4 程序设计的风格

1. 程序设计风格

程序设计风格是在程序设计时要考虑的一个重要问题,它是衡量软件设计人员程序素养的主要标志之一。良好的程序设计风格是程序设计者在长期的编程实践中逐步发展、积累和提炼出来的,它对于产生正确、高效、易读、易维护的程序是一种重要手段。

程序的风格与程序的易读性有关。如果程序设计人员养成一致的、良好的程序设计风格,则彼此之间更易于理解和交流所编写的程序。狭义地说,程序的风格涉及注解与空格的适当使用,以及尽量使助忆名具有恰当的含义,让每个语句占一行,并且明白地表示程序的结构和数据结构。

(1) 助忆名:选取具有实际意义的标识符作为变量名,可以增强程序的可读性。标识符仅仅是用来表示常量、类型、标量、函数等名称的一个记号,从语法上讲,是没有固定含义的,但是从使用上讲,每个被标记的对象都是有具体含义的。为了帮助记忆,应该尽量使用和标记对象的数学物理含义相一致的符号作为其名称。这样,有助于记忆这些标识符所标记的对象,有利于阅读、理解和修改程序。通常,用对象的英语单词或汉语拼音作为表示该对象的标识符。

(2) 注解:在程序中适当加一些注解,有助于理解该程序的功能。某个变量或常量的实际含义,某个算法的定义或名称,程序各个段落的划分,程序编写者的姓名,编程时间以及修改版号等,都可加以注解。这样,可以为编程者和使用者提供了许多附加信息,以利于日后阅读和理解该程序。通常的做法是,安排整体性的注解和局部性的注解相结合。前者包括在程序和函数的开始部分插入注解,给出相关信息。后者包括为每个被说明的对象给出注解以及在程序段落之间给予说明等。

(3) 缩进规则和空白的使用:为了增加程序的可读性以及便于对程序进行检查、修改,应养成按一定的缩进规则书写程序和在程序中使用空白的良好习惯。随着程序结构的复杂化和编码数目的大幅度增加,按照缩进规则书写的程序看起来错落有致,易读性好。空白包括空白行和一行中一些空格产生的空白区。前者可以划分一个程序的不同段落,使整个程序段落分明,意义清楚。后者可以使文字、数据、符号之间易于辨认,提高程序的可读性。

(4) 交互性:在输入语句的前面配上一个输出语句,显示适当的文字信息,用以提示程序员应当如何操作,使得在操作者与计算机之间建立了一种友好的、交互式的联系。这种交互性使得生手运行该程序时,也可以根据所提示的信息键入正确的内容。

(5) 结果输出:这是程序设计中的一个重要环节,恰当的安排输出格式,可使输出的结果直观、完整、便于查阅和存档。

(6) 常量使用:程序中有的常量,往往在多处被使用。可引入一个标识符作为它的同义词。这样的优点是易读易改,便于程序调试。

程序设计是一项人工活动,但又是一种现代产业形式中的一环。在计算机系统发展的早期时代,通用硬件相当普遍,软件却是为每个具体的应用而专门编写的。当时的软件通常是规模较小的程序,编写者与使用者常常是同一个人。当时的软件设计通常是在设计者头脑中进行的一个隐含的过程,除了程序清单以外,没有其他文档资料保存下来。这种个体化的软件环境,使得软件设计人员不大注意程序的可读性和可操作性。

2. 形成良好的程序设计风格

树立良好的程序设计风格,将能产生一个个有效、适用、易懂、易操作的程序。下面是一些建议原则,程序设计者注意照此办理,养成习惯,便可以形成良好的程序设计风格。

(1) 力求程序清晰易读。

(2) 输入尽可能简单方便,输出便于存档。

(3) 适当使用注解,使程序自成文档。

(4) 选用助忆名作为变量名。

(5) 尽量使用符号常量。

(6) 加括号与空格以避免混淆。

(7) 保持程序的交互性,使程序易于操作。

(8) 发挥计算机高速大容量的特点,使用简单的数据结构和算法,让计算机多做工作。

(9) 采用模块化方法设计程序。

(10) 使用结构化编码技术写程序。

(11) 分模块调试较大的程序。

(12) 用各种可能的情况检验程序。

(13) 测试输入数据的合理性与合法性。

(14) 对误操作安排防治措施。

(15) 先保证程序的结果正确,再考虑提高效率。

1.5 现代程序设计方法

1. 现代程序设计

通过高级语言程序设计的学习,大多数读者已经能够编写较小规模的程序,解决一定范围内的应用问题。

然而,随着科学技术的飞速发展,需要解决的问题规模越来越大。例如,美国前后四代宇宙飞船的软件规模呈指数增长,20 世纪 70 年代末穿梭号宇宙飞船的软件包含 4000 万行目标代码。假设一个人一年可以开发出一个一万行的程序,为了开发一个 4000 万行的软件,仅仅集中 4000 个程序设计者是远远不够的。程序代码长度增加了 4000 倍,程序

复杂程度的增加远远超过了4000倍。为了使许多人完成的工作合在一起确实能构成一个高质量的大型软件，不仅涉及许多技术问题，更重要的是必须有严格科学的管理。这是软件工程所讨论的问题，我们在此不做详细的讨论。本书只在具有软件工程思想的条件下，讨论实用软件设计方法与技术。

现代软件产业，需要从业人员掌握一套比较规范化、工程化的程序设计方法与技术，以及养成良好的程序设计风格与习惯，这正是本书所主要涉及的问题。否则，很可能花费了大量的人力、物力和时间，得到的却是一个潜伏着各种危险因素的不可靠的程序，而要从一个大规模的程序中发现潜在的错误，是十分困难的。

软件不同于硬件，它是计算机系统中的逻辑部件。在写出程序代码并在计算机上试验运行之前，软件开发过程中的进展情况较难判断，软件开发的质量也较难评价，因此，管理和控制软件开发过程是相当困难的。正确的程序设计方法与技术显得尤为重要。

2. 程序的结构化

结构化程序设计方法强调从程序的结构和风格上来研究程序设计。结构化程序设计，简单地说，就是按照一组能够提高程序易读性与易维护性的规则进行程序设计的方法。

它不仅要求所编写的程序结构良好，还要求程序设计的过程也是结构良好的。

结构良好的程序具有以下特征：

(1) 具有单入口、单出口的性质；

(2) 程序的执行时间是有限的；

(3) 程序中所有的语句都有被执行的机会。

这种程序在结构上应该符合以下要求：

(1) 全部程序是由一个个模块构成的；

(2) 每个模块由且仅由三种基本结构：顺序结构、选择结构、循环结构组成；

(3) 每一种基本结构都具有上述三个特征。

可以想象，这样设计出来的程序，实际上是由许多语句串联起来的顺序结构，就像项链是由一串珍珠串联起来的一样。由于每个语句都只有一个入口和一个出口，所以程序的来龙去脉清楚、段落层次分明，模块组装合理，即使程序较长，仍然具有较好的可读性与易维护性。

程序的结构性首先表现在表达式的计算中。高级语言允许使用普通代数形式来书写表达式，使得一组操作可以汇合在一个表达式中得以体现。

其次是数据的结构化。在C－PASCAL系列语言中，具有构造类型数据，即结构化的数据。一个构造类型数据的每一个成分都是一个变量，这个变量可以是非构造类型，也可以是构造类型，从而可以形成较为复杂的数据类型，例如：数组的数组、结构体数组、指针数组、链表等，能够适应不同领域应用问题的需要。

再次是语句编码的结构化。因为程序是由一系列实现规定操作的语句编码组成的，所以需要改进语句编码本身，使其具有结构化的特点。结构化的语句编码体现了能用有限数量的基本结构来表示程序控制逻辑的技术。结构化程序设计中使用的三种基本结构：顺序结构、选择结构、循环结构以及子程序模块结构，不仅可使程序编码相对集中，更

重要的是它提供了构造一个具有层次结构的程序的有效方法。这样编写的程序，增加了可读性，改善了可靠性，便于程序的修改与维护。

最后是设计与编写程序所使用的方法。结构化程序设计技术所提供的模块化、自顶向下、逐步精化的程序设计方法，是设计良好程序的有效方法。

3. 模块化程序设计

众所周知，计算机硬件的发展，经历了由电子管、晶体管、集成电路到超大规模集成电路的几个发展阶段。所有从事电子线路工作的人，都知道集成电路优于分立元件。一个由分立元件构成的电子设备是很难装配与维修的，而由集成电路组装起来的电子仪器则方便得多。

同样，计算机软件的发展，也使得程序设计逐步由许多语句编码构成一个程序，发展到由若干个模块组装成一个大的程序。这是现代化大生产的标准化、规模化、集约化的必然结果，作为现代化产业之一的软件工程也不例外。

为了得到结构良好的程序，通常采用分解的方法来对付复杂问题。对于一个复杂的问题，不是直接用一个一个的语句编码来编写程序，而是先把一个大而复杂的问题分解成若干个功能比较单纯的小问题。即将一个需要求解的问题视为一项任务，首先将这项任务映射成一个相应的体系结构，该结构中包含若干个意义明确的、相对独立的子任务。当每一个子任务获得解决，整个任务便宣告完成。对于这些子任务来说，各自涉及的范围已较原来的总任务缩小，职能也更加具体，所需的语句编码也比实现整个任务小了许多。当然，其中还可能有较为复杂的子任务。我们可以进一步将其分解成问题范围更小、职能更趋具体、所需语句编码更少的若干个子子任务。如此继续下去，直到每一个细分了的子子任务均可用简单明确的语句编码满意地实现时为止。

例 1.1　平面图形的几何变换。

[问题分析]

在计算机绘图系统中，常常需要把一个平面图形平移到另一个位置，或者改变图形的大小与形状，这种图形处理称为平面图形的几何变换。由于平面图形可以视为许多点组成的集合，因此，对平面图形进行的变换也就是对集合内各个点进行的变换。

常用的将平面上一点(x,y)变换成点(x′,y′)的几何变换，可以用矩阵形式的变换方程表示。

几何变换的变换矩阵 B 如下：

(1) 平移变换：

b11 = b22 = b33 = 1,

b31 = dx,

b32 = dy,

b12 = b13 = b21 = b23 = 0。

(2) 比例变换：

b11 = sx,

b22 = sy,

b33 = 1,

b12 = b13 = b23 = b21 = b31 = b32 = 0。

(3) 旋转变换:

b11 = b22 = cos(t),

b12 = - b21 = sin(t),

b33 = 1,

b13 = b23 = b31 = b32 = 0。

(4) 对称变换:

对 x 轴:

b11 = 1,

b22 = - 1,

b33 = 1,

b12 = b13 = b23 = b21 = b31 = b32 = 0。

对 y 轴:

b11 = - 1,

b22 = 1,

b33 = 1,

b12 = b13 = b23 = b21 = b31 = b32 = 0。

对原点:

b11 = - 1,

b22 = - 1,

b33 = 1,

b12 = b13 = b23 = b21 = b31 = b32 = 0。

对 45°直线:

b12 = - 1,

b21 = - 1,

b33 = 1,

b11 = b13 = b23 = b22 = b31 = b32 = 0。

[程序设计]

第一步:任务分解。

首先将总任务分解成若干个子任务。

平面图形的几何变换程序由以下模块组成:

(1) 主函数模块;

(2) 坐标变换函数模块;

(3) 几何变换函数模块;

(4) 坐标轴函数模块;

(5) 点变换函数模块;

(6) 画图函数模块。

第二步:总体设计。

整个程序的结构为：

坐标变换模块；

几何变换模块；

坐标轴变换模块；

点变换模块；

画图模块；

主函数模块。

第三步：构造各个模块。

主函数模块首先输出提示，请用户进行选择。根据用户输入的选择，实现相应的变换：

输入：p，　　变换：平移；

s，　　比例；

r，　　旋转；

d，　　对称；

q，　　退出。

坐标变换函数模块包括：

坐标变换函数：

scx ( )；

scy ( )；

scx1 ( )；

scy1 ( )。

分别将输入点的坐标由屏幕坐标系变换到通常的世界坐标系。

几何变换模块包括：

平移变换函数：

para ( )；

旋转变换函数：

rotate ( )；

rotate1 ( )；

比例变换函数：

scale ( )；

scale1 ( )；

对称变换函数：

dco ( )；

dcx ( )；

dcy ( )；

dcxy ( )。

分别完成相应的几何变换。

坐标轴变换模块包括：

坐标轴变换函数：

axis ( )；

axis1 ( )。

进行坐标轴从屏幕坐标系到世界坐标系的变换。

点变换模块包括：

点变换函数：

afx ( )；

afy ( )。

完成相应的变换。

画图模块包括：

顶点计算函数：

cxy ( )；

计算四边形的五个顶点坐标。

绘制图形函数：

gra ( )；

gra1 ( )。

由给出的顶点坐标，绘制两个四边形。

第四步：构造程序框架。

整个程序的总体框架为：

```
/* 坐标变换模块 */
int scx ( float xj )
{
}
int scy ( float yj )
{
}
int scx1 ( float xj )
{
}
int scy1 ( float yj )
{
}
/* 几何变换模块 */
void para ( float dx, float dy )
{
}
void rotate ( float thera )
{
```

```
}
void rotate1 (float thera )
{
}
void scale ( float s )
{
}
void scale1 (float x, float y, float s )
{
}
void dco ( float aa, float bb, float cc )
{
}
void dcx ( )
{
}
void dcy ( )
{
}
void dcxy ( )
{
}
/* 坐标轴变换模块 */
void axis ( )
{
}
void axis1 ( )
{
}
/* 点变换模块 */
float afx ( float x, float y, float d )
{
}
float afy ( float x, float y, float d )
{
}
/* 画图模块 */
void cxy ( float x1[], float y1[], float x2[], float y2[] )
```