# 第1章 用C++来解决问题

**本章提要**

在本章，我们将会介绍针对一个问题提出计算机解决方案需要做哪些事。首先，我们可能需要用一到两个段落来做一下问题的描述。然后，对于从理解这个问题的描述到具体实现一个可行的计算机解决方案的这个过程，我们就称之为解决问题。接下来，我们将在本章学习以下内容：

* 示范解决问题的过程。
* 介绍算法的特征。
* 介绍如何利用算法模式来辅助程序设计。
* 介绍类与其多个实体对象之间的关系，以及这些对象的名称、状态和操作集。
* 介绍在软件开发的实现阶段中可能出现的错误分类。

## 1.1 解决问题

解决问题的方法有很多种。在本章，我们首先要研究的是一个三步走策略，即分析、设计、实现策略。

|  |  |
| --- | --- |
| 步骤 | 具体活动 |
| **分析** | 理解待解决问题。 |
| **设计** | 根据解决方案的概要设计出算法。 |
| **实现** | 写出可执行程序的代码。 |

接下来，我们将通过一个“计算课程成绩”的示例来逐一示范这个三个步走策略中的各个步骤，看看它们在解决问题过程中所发挥的作用，并以此开始《计算基础》这门课的学习。

### 1.1.1 分析（提问、考察、研究）

程序的开发通常始于针对某个问题的研究或分析。这是很显然的，如果我们想要确定一个程序要执行哪些操作，当然先得理解该程序要解决的问题。如果该问题已经完成了书面化描述，我们就可以从阅读这个问题开始进入分析步骤了。

在分析一个问题的过程中，做好对程序所需信息数据的命名工作会是很有帮助的。例如，我们可能会被要求计算出特定飞机在特定气象条件（譬如温度、风向等）下，在指定机场跑到上可以成功起飞时的最大重量。这时，我们就可以在分析问题时将这项要计算的信息命名为maximumWeight，并将计算该信息所需的信息命名为temperature 、windDirection等。

虽然这些数据并不代表整个解决方案，但它们的确表述了问题的某个重要部分。这些数据名称会是我们编写程序，以及在程序中进行计算工作时要用到的符号，譬如可能我们要计算的是飞机在temperature的值为19.0时的maximumWeight。总而言之，这些数据通常都要经过各种形式的操作或处理之后，才能得到我们所期待的结果。在这其中，有些数据得从用户那里获取，也有些数据得经过一些相乘或相加的运算，还有些数据得在计算机屏幕上显示。

在某些时候，这些数据的值会被存储在计算机的内存中。当程序运行时，相同内存位置上的值是会变化的。另外，这些数据值通常都会有一个类型，譬如整数类型、浮点数类型、字符串类型或其他各种存储类型。对于这种用于在程序运行时存储这些可变值的内存区块，我们称之为*变量*。

我们将会看到这些数据值施以某种特定行为意义的操作，这些特定的意义有助于我们将这些数据区分成由计算机显示的数据——*输出*，和计算出结果所需的数据——*输入*。这些变量帮我们总结出了一个程序必须得做的事情。

* **输入：** 用户在解决问题过程中必须提供的信息。
* **输出：** 计算机必须显示的信息。

通常情况下，我们都可以通过回答“给定输入能得到什么输出？”这个题目来更好地理解自己要解决的问题。因此，针对待解决的问题来进行举例往往是个不错的思路。下面就是两个通过变量名的选择来精准描述其存储值的问题：

|  |  |  |  |
| --- | --- | --- | --- |
| 待解决的问题 | 变量 | 输入/输出 | 问题样例 |
| 每月还贷计算 | amount  rate  months  payment | 输入  输入  输入  输出 | 12500.00  0.08  48  303.14 |
| 计算莎士比亚的某指定剧本中某特定词的出现次数 | theWork  theWord  howOften | 输入  输入  输出 | 《无事生非》 thee  74 |

现在来总结一下，我们在分析问题过程中需要：

1. 阅读并理解待解决问题的书面说明。
2. 定义用来表示问题答案的数据，以作为输出。
3. 定义用户为获取问题答案必须要键入的数据，以作为输入。
4. 创建一些问题样例，以作汇总之用（就像上面做的那样）。

当然，教科书上的问题有时会提供清楚的变量名，以及输入/输出时用到的值类型（譬如字符串、整数、浮点数等）。如果没有的话，它们识别起来也往往是相对比较容易的。但在现实中，对于相当规模的问题来说，分析问题这个步骤通常是需要花费大量精力的。

**自检题**

1-1. 请针对英镑与美元之间的汇率转换问题，分别为用于存储用户输入值，以及程序输出值的变量赋予有意义的命名。

1-2. 请针对“从拥有200张CD的播放器中选取一张CD来播放”这个问题，分别设定用来表示所有CD，以及表示用户所选择的那张CD的变量名。

|  |  |  |
| --- | --- | --- |
| **问题分析示例** | | |
| *问题：*请根据右侧的课程成绩估算表，用作业项目、期中考试和期末考试这三项的加权值计算出这一门课的成绩。 | 参考项 | 权重比 |
| 作业项目 | 50% |
| 期中考试 | 20% |
| 期末考试 | 30% |

如前所述，问题分析的工作要从理解问题的书面描述开始，然后确定解决该问题所需要的输入和输出。在这里，先定义并命名好输出的内容是一个不错的切入点。因为，输出内容中通常存储的就是这个待解决问题的答案，它会驱使我们去深入理解这个待解决的问题。

一旦我们定义好了解决问题所需的数据，并赋予它们有意义的变量名之后，就可以将注意力转向如何完成任务了。就这个特定的问题而言，它要输出应该就是实际的课程成绩，我们将这个要输出给用户的信息命名为courseGrade。然后为了让这个问题更具有通用性，我们要让用户自己键入产生计算结果所需的值。毕竟如果这个程序可以要求用户提供所需的数据，那它以后就可以用来计算多位学生任何一门课程的成绩了。在这里，我们将需要用户输入的这些数据命名为：projects、midterm和finalExam。这样一来，我们目前就已经完成了问题分析这一步骤中的前三个动作：

1. 理解待解决的问题。
2. 定义要输出的信息：courseGrade。
3. 定义要输入的数据：projects、midterm和finalExam。

接下来就需要有一个问题样例，它有助于我们创建一个测试用例（test case），以验证输入的数据和程序产生的输出结果。例如，当projects为74.0、midterm为79.0、finalExam为84.0时，其平均加权值应该为78.0：

(0.50 × projects) + (0.20 × midterm) + (0.30 × finalExam)  
 (0.5 × 74.0) + (0.2 × 79.0) + (0.30 × 84.0)  
 37.0 + 15.8 + 25.2  
 78.0

到这里，问题的分析步骤就算完成了，我们确定了用于输入/输出的变量，这有助于我们了解计算机解决方案需要做哪些事，同时还获得了一个现成的测试用例。

|  |  |  |  |
| --- | --- | --- | --- |
| 待解决的问题 | 变量 | 输入/输出 | 测试用例 |
| 计算某门课的成绩 | projects | 输入 | 74.0 |
| midterm | 输入 | 79.0 |
| finalExam | 输入 | 84.0 |
| courseGrade | 输出 | 78.0 |

**自检题**

1-3. 请完成对下面问题的分析，这里你可能会需要用到一个准确的计算器。

*问题：* 请针对某项投资的当前价值、投资期限（可能以年为单位）以及投资利率，估算出它的未来价值。在这里，投资利率和投资期限是步调一致的。也就是说，如果投资期限以年为单位，那这里的投资利率就是年利率（例如8.5%，就是0.085）；如果投资期限以月为单位，那这里的投资利率就是月利率（例如，如果年利率是9%，那月利率就是0.075）。其未来价值的计算公式如下：

futureValue = presentValue \* (1 + rate)periods

### 1.1.2 设计（模型、思考、计划、策划、模式、纲要）

*设计*这个概念背后说代表的是一系列动作，这其中包括为程序中的每个组件安排具有针对性的算法。而*算法*则是指我们在解决问题或达成某项目标的过程中所要完成的逐个步骤。一个好的算法必须要：

* 列出程序所要执行的动作。
* 按照恰当的顺序列出这些动作。

事实上，我们可以烤制胡萝卜蛋糕的过程看成是一个算法：

* 将烤箱预热至350°F。
* 将每个烤模的侧面和底部抹上油。
* 将食材放到一大碗里进行搅拌。
* 将搅拌物倒入每个烤模中，并立即放入烤箱烤制。对于纸杯蛋糕，请倒至2/3满即可。
* 根据相关图表来进行烤制。
* 当将牙签插入到蛋糕中心，拔出来后依然能干净就表示蛋糕烤制成功。

如果这些步骤的顺序被改变了，厨师可能得到的就是一个滚烫的烤模，里面放了一团鸡蛋与面粉的搅拌物。如果省去了其中的某一个步骤，厨师也不会烤成蛋糕，或许他只是点了一次火而已。当然，熟练的厨师通常是不需要这种算法的。但是，蛋糕制作原料的销售商可不能，也不该假设他们的客户都很熟练。总之，好的算法必须要按照恰当的顺序列出恰当的步骤，并且要详尽到足以完成任务。

**自检题**

1-4. 烤制蛋糕的食谱通常会省略一个非常重要的动作，请指出上述算法中缺少的是什么动作。

通常情况下，算法中所包含的都是一些不涉及太多细节的步骤。例如，“在大碗中搅拌”这并不是一个非常具体的动作描述，里面的食材配比是什么呢？如果我们现在的问题是要编写一个人类能够理解的蛋糕烤制算法，这个步骤就可以做进一步的改进，使其能指导厨师更好地安排食材配比。譬如我们可以将该步骤改成“将牛奶倒入盛有鸡蛋与面粉的大碗中搅拌，直至其表面光滑”，或者为面包师将该步骤切分如下：

* 配置好食材中干燥的成分。
* 将食材的液体成分倒入碗中。
* 每次倒入四分之一杯的干成分，将其搅拌至表面光滑。

算法可以用*伪代码*来描述，甚至也可以用一种非程序员也能理解的语言来描述。由于伪代码面向的是人类，而不是计算机，所以用伪代码描述的算法在程序设计中是很有帮助的。

伪代码有极强的表达能力。一条伪代码通常可以表示多条计算机指令。另外，用伪代码来描述算法可以避免纠缠于标点错误或者与特定计算机系统相关的细节。用伪代码来描述解决方案允许我们将这些细节问题向后推，这可以让设计变得更容易一些。其实，写算法就相当于在做计划，程序开发者也可以用纸和笔来做这些设计，甚至有时可以直接在脑海中完成这些事。

### 1.1.3 算法模式

解决问题通常需要用户完成一定的输入才能计算并显示出相应的信息。事实上，这种输入-处理-输出的动作流是如此的司空见惯，我们甚至可以把它视为一种模式，而且你们会发现这绝对是程序设计中最有用的几个算法模式之一。

*模式*可以是任何一种事物形式或设计，它的作用是将某些事物模型化或者提供某种行事指南。而算法模式就是一种用于辅助我们解决问题的指南。以下面的输入/处理/输出（即Input/Process/Output，简称IPO）算法模式为例，我们可以用它来辅助解决第一个问题的设计，事实上，IPO模式可以辅助我们解决本书前五章中几乎所有程序的设计问题。

|  |  |
| --- | --- |
| 算法模式 | *输入/处理/输出* |
| 模式： | 输入/处理/输出（IPO） |
| 问题： | 程序需要基于用户的输入来计算并显示我们所需的信息。 |
| 纲要： | 1. 获取输入数据。 2. 用某种有意义的方式处理数据。 3. 输出结果。 |

**代码示例：**

int n1, n2, n3;  
oat average;  
  
// Input  
cout << "Enter three numbers: ";  
cin >> n1 >> n2 >> n3;  
  
// Process  
average = (n1 + n2 + n3) / 3.0;  
  
// Output  
cout << "Average = " << average;

这是若干种算法模式中的第一种。在后面的章节中，我们会陆续看到Guarded Action、Alternative Action、Indeterminate Loop等其他算法模式。为了有效地使用一个算法模式，我们首先必须得熟记它。将IPO模式注册在心中，并在开发程序时能想起它，这样就会让我们的程序设计变得更容易。例如，如果你在数据数据中发现了无意义的值，有可能是你将程序的处理步骤放在了输入步骤*之前*，或者根本就跳过了输入步骤。

关于模式在解决其他类型问题时所能提供的帮助，我们可以参考Christopher Alexander在他*A Pattern Language*

这本书里的这样一段话：

每个模式描述的都是一个我们所在客观环境中反复出现的问题，及其解决方案的核心内容，通过这种方式构建的解决方案，可以让我们用上一百万次，无须用相同的方式构建两次解决方案。

尽管Alexander所描述的是用于设计家具、花园、大楼和城镇的模式，但他描述的模式也适用于计算领域问题的解决。在程序设计的过程中，IPO模式就是会反复出现，并指引着许多问题的解决方案。

### 1.1.4 算法设计示例

IPO模式也可以用来指导我们解决之前那个课程成绩计算问题的算法设计：

|  |  |
| --- | --- |
| 三步骤模式 | 将模式应用到具体的算法中 |
| 1. 输入 | 1. 读取projects、midterm和finalExam三个变量 |
| 2. 处理 | 2. 计算出courseGrade的值 |
| 3. 输出 | 3. 显示courseGrade的值 |

当然了，算法的开发通常是一个迭代的过程，模式也只是提供了解决这个问题所必需的动作序列纲要。

**自检题**

1-5. 在阅读上述算法的三个动作时，你发现其中缺失的动作了吗？

1-6. 在阅读上述算法的三个动作时，你有发现什么不正常的动作吗？

1-7. 如果对调上述算法中前两个动作的顺序，该算法还能正常工作吗？

1-8. 上述算法的描述是否已经足够支持计算出courseGrade的值了？

很显然，我们在上面对计算课程成绩问题的的处理步骤的描述是不够详细的，我们还需对它进行进一步的细化。具体来说就是，说清楚在处理过程中如何用输入数据计算出课程成绩。上面的算法中省略了我们在问题书面化描述中提到的加权值，所以在下面，我们在第二步中重新细化了处理步骤：

1. 从用户那里获取projects、midterm、finalExam这三个数据值。
2. 计算courseGrade = (projects × 50%) + (midterm × 20%) + (finalExam × 30%)。
3. 显示courseGrade的值。

就像人们常说的那样，好的艺术家应该知道什么时候该放下画笔，并决定与此刻完成他的画作，这对他的成功是至关重要的。同样地，设计师也必须要知道什么时候该停止设计，那就是我们进入解决问题第三阶段——实现阶段的好时机。

现在，我们来总结一下到目前为止所取得的进展：

* 待解决的问题得到了充分的理解。
* 所要用到的变量得到了确认。
* 对已知问题样例的输出有了了解（78.0%）。
* 已经开发出了一种算法。

### 1.1.5 实现（完成、操作、使用）

计算机本质上就是一种可编程的、用来存储、检索并处理数据的电子设备。事实上，程序员们也可以通过纸和笔来手动执行存储、检索与处理数据的动作，以此来模拟算法在电子设备中的执行过程 。下面就是一个人工模拟的（即非电子的）算法执行过程：

1. 从用户那里检索到一些示例值并将它们存储起来：

projects = 80  
 midterm = 90  
 finalExam = 100

1. 再次检索这些值并用它们计算出courseGrade的值：

courseGrade = (0.5 × projects) + (0.2 × midterm) + (0.3 × nalExam)  
 (0.5 × 80.0) + (0.2 × 90.0) + (0.3 × 100.0)  
 40.0 + 18.0 + 30.0  
 courseGrade = 88.0

1. 将存储在courseGrade中的值显示成88% 。

### 1.1.6 一段C++程序

下面，我们要带你来预览一段完整的C++程序，由于对这里的许多编程语言上的细节问题，我们要到下一章中才会介绍，所以各位也不必期待自己能完全理解这段C++源代码。在此次此刻，我们只需要读懂这段源代码是对之前那个伪代码算法的实现就可以了。这里有projects、midterm、finalExam这三个变量，代表的是用户的输入。另外还有一个名为courseGrade的输出变量。另外，这里的cout对象，发音是“see-out”，代表的是*公共输出*以及程序所产生的输出。而输入部分用的则是cin对象，发音是“see-in”，代表的是*公共输入*。

/\*  
 \* This program computes and displays a final course grade as a  
 \* weighted average after the user enters the appropriate input.  
 \*  
 \* File name: CourseGrade.cpp  
 \*/  
  
#include <iostream> // for cin and cout  
#include <string> // for string  
using namespace std; // avoid writing std::cin std::cout std::string  
  
int main() {  
 // Explain what this program does.  
 cout << "This program computes a weighted course grade." << endl;  
  
 // Read in a string  
 cout << "Enter the student's name: "`;  
 string name;  
 cin >> name;  
  
 // I)nput projects, midterm, and nalExam  
 double projects, midterm, finalExam;  
  
 cout << "Enter project score: ";  
 cin >> projects;  
  
 cout << "Enter midterm: ";  
 cin >> midterm;  
  
 cout << "Enter finalexam: ";  
 cin >> finalExam;  
  
 // P)rocess  
 double courseGrade = (0.5 \* projects) +  
 (0.2 \* midterm) +  
 (0.3 \* finalExam);  
  
 // O)utput the results  
 cout << name << "'s grade: " << courseGrade << "%" << endl;  
}

**会话**

下面是该程序计算一次加权课程成绩的过程：

Enter the student's name: Dakota  
Enter project score: 80  
Enter midterm: 90  
Enter final exam: 100  
Dakota's grade: 88%

### 1.1.7 测试

测试这个重要的过程，可能，可以，并且也应该出现在我们解决问题的所有阶段中。这部分的实际工作量很小，但很值得做。只不过，在因为*不做*测试而遇到问题之前，你可能不会同意我这个观点。总而言之，测试相关的系列动作可以出现在程序开发的所有阶段中：

* 在分析阶段中，我们可以通过测试用例确认自己对待解决问题的理解。
* 在设计阶段中，我们可以通过测试算法来确定其按照恰当的顺序执行了恰当的步骤。
* 在测试过程中，我们可以用几组不同的输入数据来运行程序，确认其结果是否正确。
* 请复查待解决问题的书面描述，确认我们运行的程序的确执行了需要执行的操作。

我们应该在针对问题编写程序之前（而不是之后）准备一个以上的测试用例，然后确定一下程序的输入值与预估输出值。譬如，之前我们提到的输入值为80、90和100时，预估输出值是88%的情况，就属于这样的测试用例。当程序最终产生它的输出时，我们可以拿自己预估的结果与程序实际运行中的输出进行比对，如果预期输出与程序输出对不上，我们就要及时作出相关的调整，因为这种冲突表示该问题示例或程序输出有错，甚至有可能是两者都错了。

通过若干个测试用例的测试，我们可以有效地避免误认为只要程序能成功运行并产生输出，程序就是正确的。显然，输出本身也可能会出错！简单执行一下程序是无法确保程序正确的。测试用例的作用是确保程序的可行性。

然而，即使进行了详尽的测试，我们其实也未必能完全保证程序的正确性。E. W. Dijkstra就曾认为：测试只能证明程序中存在错误，无法证明其中不存在错误。毕竟，即使程序的输出是正确的，该程序本身也未必就一定正确。但测试还是有助于减少错误，并提高程序的可信度。

**自检题**

1-9. 如果程序员预估当上述程序的三个输入都为100.0时，courseGrade的值也应为100.0，但程序显示的courseGrade的值却为75.0，请问是预估输出和程序输出中的哪一方出错了？还是双方都错了？

1-10. 如果程序员预估当上述程序的输入projects为80.0、midterm为90.0、finalExam为100.0时，courseGrade的值应为90.0，但程序显示的courseGrade的值却为88.0，请问是预估输出和程序输出中的那一方出错了？还是双方都错了？

1-11. 如果程序员预估当上述程序的输入projects为80.0、midterm为90.0、finalExam为100.0时，courseGrade的值应为88.0，但程序显示的courseGrade的值却为90.0，请问是预估输出和程序输出中的哪一方出错了？还是双方都错了？

## 1.2 对象、类型与变量

为了让输入的内容在程序中发挥作用，我们必须要在计算机内存中开辟一块“空间”来存储它们。关于这一点，C++之父Bjarne Stroustrup是这样说的：

我们将这样的一块“空间”称之为一个对象。换而言之，对象就是内存中一块带有类型信息的区域，其类型规定的是这块“空间”内所能存储的信息种类，而被命名了的对象就叫做变量。例如，字符串要放在string变量中，整数要放在int变量中。大家可以将对象看作一个“盒子”，我们可以用它来存放该对象类型的值。

例如在之前的程序中，我们就是用int类型来存储数字或整数的。在int变量上，我们可以执行包括加、减、乘、除在内的一系列操作。另外，这里需要提醒一下，C++中的乘法运算符是\*（因为用x可能会带来某种混淆）。

double courseGrade = 0.5\*projects + 0.2\*midterm + 0.5\*finalExam;

float和double这两个类型存储的是带有小数部分的数值（double是两倍大的float类型）。另外，C++的string类型中存储的是“Firstname I. Lastname”这样的字符序列，以及一个记录该字符串中字符数的整数。

*对象*是存在于计算机内存中的实体，我们可以通过一个对象所存储的值类型（即它的*属性*）以及它所能执行的操作（即它的*行为*）

来理解这个对象。也就是说，每个对象都应该有：

* 一个用于存储和检索该对象值的名称。
* 一组存储与计算机内存中的值，它们代表了该对象的状态。
* 一组该对象可以执行的操作，譬如加法、输入、输出、赋值等。

关于对象的名称、状态和操作这三个特征，我们在之前的课程成绩程序中其实都有说明。该程序用projects、midterm、finalExam这三个数字对象存储了来自键盘的输入。这些对象各自都存储了一个像79或90这样的整数[[1]](#footnote-0)。并且这些对象可以执行输入、乘法和加法操作，以此计算出了courseGrade的值。另外，这些数字对象的还用赋值操作完成了存储动作用，用cout <<操作完成了输出动作，这样用户才能看到程序处理的结果。

**首个程序中的对象特征**

|  |  |
| --- | --- |
| 名称 | 四个数字对象各自都有一个属于自己的名称做标识，譬如其中的第一个对象名为projects。 |
| 状态 | projects的值是通过cin >>输入操作来设置的，而courseGrade的状态则是通过赋值操作（使用=操作符）来定义的。最后，courseGrade的状态又是在执行cout输出操作的过程中被检索的。 |
| 操作 | int对象上还可以执行加法（+）与乘法（\*）这些其他操作。[[2]](#footnote-1) |

在C++中，类型分为基本类型和复合类型两种。其中，*基本类型*所存储的是一个固定大小的、直接与硬件对应的值，这种类型确定的是其对象中可以存储什么值，以及可以在该对象上执行什么操作。对于int和double这样的数字类型来说，其对象所占的字节数在不同的计算机中是不一样的，这决定着该对象所能存储的取值区间。

|  |  |  |
| --- | --- | --- |
| 数据类型 | 大小 | 通常情况下的取值区间（这是变化的） |
| short | 2字节（16比特） | -32768 到 32767 |
| unsigned short | 2字节 | 0 到 65,535 |
| int | 4字节 | -2147483648 到 2147483647 |
| unsigned int | 4字节 | 0 到 4294967295 |
| unsigned long | 8字节 | 0 到 18446744073709551615 |
| float | 4字节 | 3.4E +/- 38 （7位有效数字） |
| double | 8字节 | 1.7E +/- 308 （15位有效数字） |
| char | 1字节 | 0 到 255 |
| bool | 1字节 | true 或 false |

*复合类型*是一种由其他类型来定义的类型，本书将会涉及的复合类型包括：引用、函数、类、数组以及指针。举例来说，下面的string就是一个由字符和其他相关数据组成的引用类型，它可以找出某字符序列的长度，也可以从某一字符串中创建一个被指定了首尾索引的子字符串（在后续章节中，我们还会介绍更多相关的操作）：

string aString = "A sequence of characters"; // Output:  
cout << aString.length() << endl; // 24  
cout << aString.substr(2, 8) << endl; // sequence

除了string类型之外，还有两个类型我们现在就已经使用道理，它们分别是：名为cin的istream对象——它的作用是从键盘和磁盘文件这样的输入源中读取数据；以及名为cout的ostream——它的作用是输出程序产生的内容。

**自检题**

1-12. 请描述一下存储在double类型对象中的值。

1-13. 请说出两个double对象的操作名称。

1-14. 请描述一下存储在int类型对象中的值。

1-15. 请说出两个int对象的操作名称。

1-16. 请描述一下存储在string类型对象中的值。

1-17. 上面哪种类型的对象中只存储了一个值？

## 本章小结

在一章中，我们介绍了解决问题的分析、设计、实现三步走策略。下面我们用一张表来总结一下该策略的这三个阶段各自要执行的一些动作。除此之外，我们还添加了维护阶段，以补充这个三步走策略，使其成为一个完整的程序生命周期。毕竟，维护阶段的工作事实上占据了程序生命周期中大部分的时间、精力和金钱。

|  |  |
| --- | --- |
| 阶段任务 | 可执行的动作 |
| 分析 | 阅读并理解问题的书面说明。 确认要用于输入/输出的对象。 解决几个问题样例。 |
| 设计 | 找出可用于指引算法开发方向的模式。 写出一组解决问题所需要执行的算法步骤。 在具体施行该算法的过程中进一步优化它。 |
| 实现 | 将设计结果转换成编程语言。 修复其中的错误。 创建可执行的程序。 测试该程序。 |
| 维护 | 持续更新该程序，使其与时俱进。 增强该程序。 发现并纠正其中的bug。 |

* 我们还介绍了一些用于分析和设计的工具：
* 为对象起一个有助于解决问题的名称。
* 开发相应的算法。
* 优化算法中的一个或多个步骤。
* 使用输入/处理/输出模式。
* 我们还提供了示例程序，当然，我们要到下一章中才会介绍该程序中的许多细节，这里只是让读者了解一下c++中的基本类型和复合类型。
* 虽然测试很重要，但我们需要明白它不能证明程序中没有错误。当然，测试的确可以检测出部分错误，但这只能在某种程度上建构起我们对程序可行性的信心而已。

## 练习题

1. 在分析问题的阶段，我们可以执行哪些动作？
2. 一个好的算法应该具备哪些特征？
3. 用于存储输出值的对象与用于存储用户输入值的对象之间有什么差异？
4. 请列举出三个对象所具有的特征。
5. 在设计程序的阶段，我们可以执行哪些动作？
6. 怎样的设计成果是“可交付”的？
7. 该用什么类型的对象来存储注册某一门课的学生人数？
8. 该用什么类型的对象来存储π的值？
9. 该用什么类型的对象来存储一部莎士比亚戏剧的剧本？
10. 在程序开发中，实现阶段可交付的成果应该是怎样的？
11. 该如何判断一个程序的运行是否正确？请证明你的判断。
12. 请编写一个如何回到自己居住地的算法。
13. 请编写一个可在电话簿中查找任意电话号码的算法。请问该算法始终能成功找到目标吗？
14. 请编写一个能指引别人步行到你家的算法。
15. 请设法获取你系统中能用于创建、编译、连接并执行一个C++程序所需的命令，这可能需要你登入自己的系统中，找出那些可用于基本编辑和编译程序的命令。在完成这件事之后，请你编写一个完整的算法，该算法要能指导一个新手完整地编写一个能通过测试的程序，你需要列出该过程中所有必要的步骤，譬如“比对示例输入与程序输出”、“创建新文件”、“编译程序”等。

## 解决问题：请编写一个算法

### 1A 简单平均值

请编写一个算法，计算出三个权重相等的测试成绩的平均值。

### 1B 加权平均值

请编写一个能根据一下权重比计算出课程成绩的算法：

|  |  |
| --- | --- |
| 成绩评估项 | 所占权重 |
| 小考平均分 | 20% |
| 期中考试 | 20% |
| 实验成绩 | 35% |
| 期末考试 | 25% |

### 1C 批发成本

假设我们碰巧知道了商家在出售CD播放机时通常要加价25%这个信息。在这种情况下，如果CD播放机的零售价（即我们所支付的价格）是189.98美元，请问该商家进货时支付的价格（即批发价）是多少呢？或者更一般地说，我们会如何根据一个商品的零售价和商家对它的加价计算出该商品的批发价呢？请对该问题进行分析，并设计出一个能根据给定零售价和商家的加价计算出任意商品批发价的算法，你可以使用这个公式来计算批发价：$retailPrice = wholesalePrice × (1 + markup)$。

### 1D 时间差

请编写一个算法，使其能记录两列不同火车的出发时间（这里0代表凌晨零点、0700代表上午7:00、1314代表下午1:00后的第14分钟、2200代表的是晚上10点），并以小时加分钟的形式打印出这两个时间的差距。这里我们得假设双方的时间都在同一天，并且都得是有效时间。例如，1099不是一个有效时间，因为其最后两位数字代表的应该是分钟，它的取值范围应该是在00到59之间。同理，2401也不是一个有效时间，因为其前两位数字代表的是小时，它的取值范围必须在00到23之间。总之在这种情况下，如果A列车是在1255出发，而B列车则是在1305出发，那么这两列火车的时间差应该就是0小时十分钟。

1. 译者注：作者原文如此，实际上他用的是double类型的浮点数。 [↑](#footnote-ref-0)
2. 译者注：实际程序使用的是double对象，但这并不影响这里的结论。 [↑](#footnote-ref-1)