第5章 算法与程序设计基础

利用计算机求解问题的关键是对问题(需要)的描述和分析,并设计正确的算法才能够快速高效地得到正确的结果。本章主要介绍算法、程序设计方法、程序的三大基本结构以及用 Python 语言设计简单程序等知识。

5.1 认识算法

5.1.1 算法的概念

算法(Algorithm)是指解题方案的准确而完整的描述,是一系列解决问题的清晰指令,算法表示用系统的方法描述解决问题的策略机制。也就是说,能够对一定规范的输入,在有限时间内获得所要求的输出。如果一个算法有缺陷,或不适合于某个问题,执行这个算法将不会解决这个问题。完成同样的任务,不同的算法可能需要使用不同的时间、空间或效率。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。

算法中的指令描述的是一个计算,当其运行时能从一个初始状态和(可能为空的)初始输入开始,经过一系列有限而清晰定义的状态,最终产生输出并停止于一个终态。一个状态到另一个状态的转移不一定是确定的。

也可认为,算法是一组明确的、可执行的有限步骤的有序集合。“明确”是指不能模棱两可,每一步骤是必须“可执行的”。可执行不是指一定要能被计算机所直接执行,即它不一定直接对应计算机的某个指令,但至少对阅读算法的人来说,相应步骤是有效和可以实现的。例如,“列出所有自然数”就是不可执行的,因为有无穷多的自然数。

【例 5.1】 求 1+2+3+…+100 的和。

方法 1:

```
步骤 1:求 1+2 的和,得到值 3。
步骤 2:求 3+3 的和,得到值 6。
……
步骤 99:将步骤 99 的和与 100 相加得到结果 5050。
```

方法 2:

```
步骤 1:求 1+99 的和,得到值 100
```

步骤 2:求 2+88 的和,得到值 100
……
步骤 49:求 49 与 51 的和,得到值 100
步骤 50:将前面所有步骤的和加上 100,再加上 50 得到结果 5050。

上面两种解题方法是较常见的方法,但不容易扩展与描述,比较烦琐。如果要计算从 1 加到 10000 甚至更多,用上面两种方法显然是不可取的,应该找一种更通用、可描述、容易实现的算法。

对于方法 1,每一个步骤可以进行以下描述:

步骤 i:将步骤 i-1 的和 s_{i-1}(可用变量 s 表示)加 i 得到值 s_i。

在这里,用到两个变量 s 和 i,可以用循环结构的算法来求得结果,描述如下:

步骤 1:将 s 赋 0,写成 s←0;i 的值赋为 1,写成 i←1。
步骤 2:将 s 与 i 加,得到的值赋给变量 s,写成 s←s+i;i 的值增加 1,写成 i←i+1。
步骤 3:如果 i 的值小于或等于 100(i<=100),返回步骤 2,否则执行步骤 4。
步骤 4:输出 s 的值。

如果要计算 1 加到 10000,只需将步骤 3 中的“i＜100”改成“i＜10000”即可。由此可见,该算法具有可描述性、通用性和可实现性。

5.1.2 算法的特征

一个有效的算法应该具有以下 5 个特征。

1. 确定性

算法的每一步骤必须有确切的定义,每一条指令,每一个操作都是确定的,不能是模棱两可的。

2. 有穷性

算法的有穷性是指算法必须能在执行有限个步骤之后终止,算法的有穷性还包括合理执行时间的含义,例如,如果预测明天气候的算法需要执行超过 24 小时,显然就失去了实用价值。

3. 输入项

一个算法有 0 个或多个输入,以刻画运算对象的初始情况,所谓 0 个输入是指算法本身定出了初始条件。

4. 输出项

一个算法有一个或多个输出,以反映对输入数据计算后的结果。没有输出的算法是

毫无意义的。

5. 可执行性

算法中执行的任何计算步骤都是可以被分解为基本的、可执行的操作步，即每个计算步都可以在有限时间内完成(也称之为有效性)。

5.1.3 算法的描述方法

描述算法的方法有多种，常用的有自然语言、流程图、N-S 图和伪代码等，其中使用最普遍的是流程图。

1. 用自然语言描述算法

自然语言就是指用人们日常使用的语言来描述解决问题的方法和步骤，这种描述方法通俗易懂，即使不熟悉计算机语言的人也很容易理解。但是，自然语言在语法和语义上往往具有多义性，并且比较烦琐，对程序流向等描述不明了、不直观。

【例 5.2】 给出 n 个数，求最大数。

使用自然语言的算法描述如下：

```
步骤 1:假设第 1 个数是最大数 max,计数器 i 的值为 1。
步骤 2:如果 i 的值小于或等于 n,执行步骤 3,否则执行步骤 5。
步骤 3:如果第 i 个数的值大于 max,则将第 i 个数的值赋给 max。
步骤 4:i 的值增加 1。
步骤 5:输出最大值 max。
```

2. 用流程图描述算法

以特定的图形符号加上说明来表示算法的图，称为流程图或框图。流程图是流经一个系统的信息流、观点流或部件流的图形代表。常用程序图符号如图 5.1 所示。

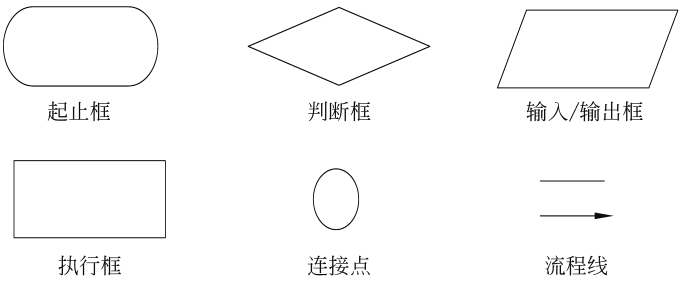

图 5.1 常用流程图符号

【例 5.3】 求 s=1+2+…+100，使用流程图描述实现算法。

使用流程图表示的算法如图 5.2 所示。

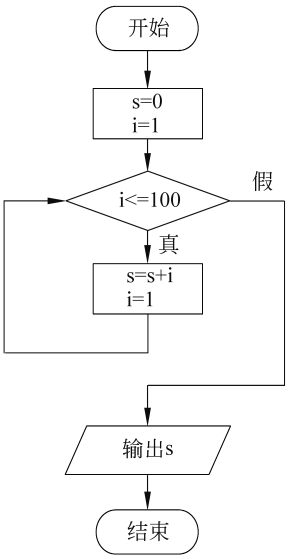

图 5.2　流程图描述算法

3. N-S 图

N-S 图，又称为盒图或 NS 图（Nassi Shneiderman 图），是结构化编程中的一种可视化建模。

1972 年，美国学者 I.Nassi 和 B.Shneiderman 提出了一种在流程图中完全去掉流程线，全部算法写在一个矩形阵内，在框内还可以包含其他框的流程图形式，即由一些基本的框组成一个大的框，这种流程图就称为 N-S 结构流程图（以两个人的名字的头一个字母组成）。N-S 图包括顺序、选择和循环三种基本结构，如图 5.3 所示。

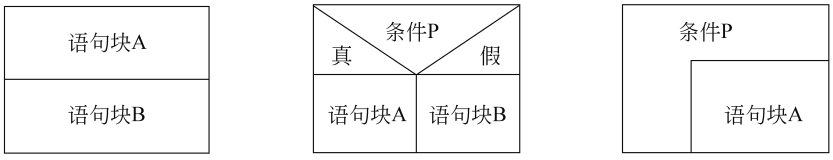

图 5.3　N-S 图的三种结构表示方法

【例 5.4】 求 s＝1＋2＋…＋100，使用 N-S 图描述实现算法。

使用 N-S 图描述的实现算法如图 5.4 所示。

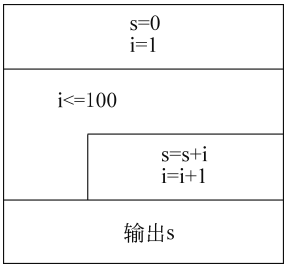

图 5.4　使用 N-S 图表示算法

4. 伪代码

伪代码是介于自然语言和计算机语言之间的文字和符号，它与一些高级编程语言（如 Visual Basic 和 Visual C++）

类似，但不需要遵循真正编写程序时的严格规则。伪代码用一种自上而下、易于阅读的方式表示算法。在程序开发期间，伪代码经常用于“规划”一个程序，然后再转换成某种语言程序。

【例 5.5】 求 s=1+2+…+100，使用伪代码表示算法。

使用伪代码表示的算法如下。

```
s=0
i=1
do while i<=100
    s=s+i
    i=i+1
end do
output s
```

5.1.4 典型算法举例

在本节中，将分别对顺序结构、选择结构和循环结构的几种结构进行举例。

1. 两数互换

两数互换的算法是顺序结构中最典型的算法，它应用在数组排序等较复杂的算法中。实现两数互换最常用的算法是使用一个临时变量来过渡，从而使两个变量中的值互换。

将变量 x 与变量 y 值互换的算法，用伪代码表示如下：

```
t=x
x=y
y=t
print x,y
```

2. 判断某个年份是否是闰年

闰年的条件是能被 4 整除且不能被 100 整除，或者能被 400 整除的年份。

能被 4 整除在数学上可表示为“与 4 求余其值为 0”，求余可用符号 mod 来表示。所以表示闰年条件的自然语言语句可换成以下伪代码语句：

```
if y mod 4=0 and y mod 100<>0 or y mod 400=0
```

在这里 y 表示年份。

使用伪代码表示如下：

```
input y
if y mod 4=0 and y mod 100<>0 or y mod 400=0 then
```

```
    print y 是闰年
else
    print y 不是闰年
endif
```

3. 循环结构算法举例

【例 5.6】 求 s=n!,使用流程图描述实现算法。

本题与求 s=1+2+…100 算法类似,只需要把 100 改为 n,n 从键盘输入,将加号改为乘号。注意,在累加中,s 的初值一般从 0 开始,在累乘中,s 的初值则需要从 1 开始。

使用流程图表示的算法如图 5.5 所示。

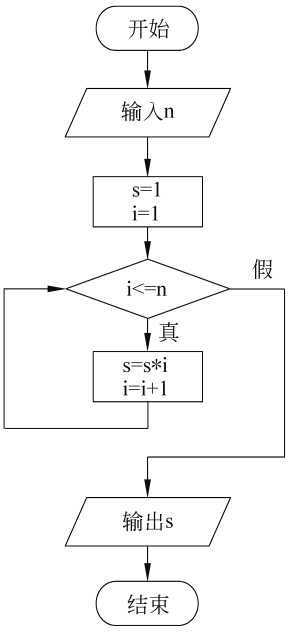

图 5.5 使用流程图表示求 n! 的算法

【例 5.7】 求 s=1!+2!+…+n!,使用伪代码描述实现算法。

在本题中,需要用到循环的嵌套,内循环为求 n!,外循环为将所求阶乘累加。

使用伪代码表示的算法如下:

```
input n
s=0
i=1
do while i<=n
    f=1 //f 为所求的 i 的阶乘
    j=1
    do while j<=i
```

```
        f=f*j
        j=j+1
    end do
    i=i+1
end do
output s
```

5.2 程序设计方法

计算机程序也就软件，软件(Software)这一名词在20世纪60年代初从国外传来，当时许多人说不清它的确切含意。有人译为“软制品”，也有人译为“软体”，现在统一称它为软件。

早期的软件开发具有随意性、无计划和不规范性，产生了像软件开发无计划、需求分析不充分、开发过程无规范、软件产品无评测手段等问题，出现了“软件危机”。人们开始重新思考程序设计的基本问题，即程序的基本组成、设计方法等。

5.2.1 程序设计过程

20世纪70年代以前，人们曾经把程序设计看作是一种任人发挥创造才能的技术领域。当时一般认为，写出的程序只要能在计算机上得出正确的结果，程序的写法可以不受任何约束。但随着程序越来越长，结构越来越复杂，就出现了很多问题，直到出现“软件危机”。

设计一个功能全面的计算机程序，一般有需求分析、程序设计、程序编码、程序测试和程序交付等5个步骤。这5个步骤在时间上从前向后进行，但没有特别严格的规定，在进行后一个步骤时，也可对前面的步骤进行修改。

1. 需求分析

需求分析是系统分析员对用户的需求与实际问题进行分析，包括已知是什么，需求是什么。此步骤解决软件要做什么，而不是软件应该怎么做。

2. 程序设计

程序设计包括概要设计和详细设计。概要设计包括系统的基本处理流程、系统的组织结构、模块划分、功能分配、接口设计、运行设计、数据结构设计和出错处理设计等，为软件的详细设计提供基础。

在概要设计的基础上，开发者需要进行软件系统的详细设计。在详细设计中，描述实现具体模块所涉及的主要算法、数据结构、类的层次结构及调用关系，需要说明软件系统各个层次中的每一个程序(每个模块或子程序)的设计考虑，以便进行编码和测试。

3. 程序编码

在程序编码阶段，开发者根据《软件系统详细设计报告》中对数据结构、算法分析和模块实现等方面的设计要求，开始具体地编写程序工作，分别实现各模块的功能，从而实现对目标系统的功能、性能、接口、界面等方面的要求。

4. 程序测试

软件在交付用户前要进行测试，程序测试的目的和任务是要发现程序中的错误。如果软件交付给用户后再发现错误，付出的代价会是在测试中发现错误的几倍甚至几十倍。一个好的测试方法是在尽量少的时间里发现更多的错误。

5. 程序交付

在程序测试达到要求后，软件开发者应向用户提交开发的目标安装程序、数据库的数据字典、《用户安装手册》《用户使用指南》《需求报告》《设计报告》《测试报告》等文档。

5.2.2 程序设计方法

程序设计方法主要有结构化程序设计方法与面向对象程序设计方法。

1. 结构化程序设计方法

结构化程序设计方法的主要特点是采用自上而下、逐步求精的程序设计方法。使用三种基本控制结构构造程序，任何程序都可由顺序、选择、循环三种基本控制结构构造。以模块化设计为中心，将待开发的软件系统划分为若干相互独立的模块，这样使每一个模块的工作变单纯而明确，为设计一些较大的软件打下良好的基础。

(1) 结构化程序设计主要原则

① 自上而下。程序设计时，应先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标。不要一开始就过多追求众多的细节，先从最上层总目标开始设计，逐步使问题具体化。

② 逐步细化。对复杂问题，应设计一些子目标作为过渡，逐步细化。也就是说，把一个较大的复杂问题分解成若干相对独立且简单的小问题。

③ 模块化。一个复杂问题，肯定是由若干简单的问题构成。模块化是把程序要解决的总目标分解为子目标，再进一步分解为具体的小目标，把每一个小目标称为一个模块。模块化的目的是降低程序复杂度，使程序设计、调试和维护等操作简单化。

一个模块可以是一段程序、一个或多个函数或过程等。

(2) 结构化程序设计的优缺点

① 优点。由于模块相互独立，因此在设计其中一个模块时，不会受到其他模块的牵连，因而可将原来较为复杂的问题化简为一系列简单模块的设计。模块的独立性还为扩充已有的系统、建立新系统带来了不少的方便，因为我们可以充分利用现有的模块作

扩展。

② 缺点。

- 通常情况下，用户对自己的需要也不是特别明确，在软件的开发过程中会发生变化。用户需求难以在系统分析阶段准确定义，致使系统在交付使用时产生许多问题。
- 用系统开发每个阶段的成果来进行控制，不能适应事物变化的要求。
- 系统的开发周期长。

所以，对于大型软件的开发，一般使用面向对象的程序设计方法。

2. 面向对象的程序设计方法

面向对象的程序设计方法（Object-Oriented Programming，OOP）是对面向过程程序设计方法（Process-Oriented Programming，POP）的继承和发展，它吸取了面向过程程序设计方法的优点，同时又考虑到计算机程序与现实世界的联系。OOP 将客观世界看成是由各种各样的实体组成的，这些实体就是 OOP 中的对象。

（1）基本概念

在面向对象的程序设计中，引入了类、对象、属性、事件和方法等一系列概念以及编程思想。

① 类。类（Class）是具有共同属性的事物的抽象。通常来说，类定义了事物的属性和它可以做到的动作（它的行为）。例如类“人”，具有共同的属性：姓名、身高、出生日期等，具有共同的行为：工作、休息等。

② 对象。对象（Object）是类的实例。类是抽象的，对象是具体的。例如对于类“人”，其实例“张三”是一个具体的人，该实例继承了类“人”的属性和行为。对于属于同一类的不同对象，其属性和行为是不同的。

在程序的运行过程中，系统为对象分配内存空间，而不会为类分配内存空间。这很好理解，类是抽象的，系统不可能给抽象的东西分配空间，而对象则是具体的。

③ 属性。类的属性是一个数据项，用于描述类的所有对象的一个共同特征，对于任何对象实例，它的属性值是相同的。不同的编程语言对属性有不同的定义，在面向对象的编程语言 Python 中，类属性就是在类中定义的变量。

④ 事件和方法。事件和方法均是对象的行为，在不同的面向对象的计算机语言中，这两者的定义有细微的差别。一般来讲，程序员需要为事件编写程序代码，否则该事件对应的行为不起任何作用，但不需要为方法编写程序代码，可以直接调用，不过，根据方法的定义格式，有时需要设置某些参数。例如，在窗体 Form1 上画一个半径为 5 厘米的圆，可用以下类似语句：

```
Form1.SetCircular(5)
```

在这里，Form1 是对象名称；SetCircular 是方法名；5 是方法的参数，单位是厘米。

⑤ 消息。消息是对象之间传递的信息，它请求对象执行某一处理或回答某一要求的信息，统一了数据流和控制流。消息是对象之间交互的唯一途径，一个对象要想使用其他

对象的服务，必须向该对象发送服务请求消息；而接收服务请求的对象必须对请求做出响应。

(2) 面向对象程序设计特征

面向对象程序设计(OOP)是以数据为中心，将数据和处理相结合的一种方法，这种方法具有以下 3 个基本特征。

① 封装性。封装性是指将对象相关的信息和行为状态捆绑成一个单元，即将对象封装为一个具体的类。封装隐藏了对象的具体实现，当要操纵对象时，只需调用其中的方法，而不用管方法的具体实现。

② 继承性。一个类继承自另一个类，继承类可以获得被继承类的所有方法和属性，并且在继承类中可以根据实际的需要添加新的方法或者对被继承类中的方法进行覆盖，被继承类称为父类或者超类。继承类称为子类或导出类。继承提高了程序代码的可重用性。例如，Java 语言中一个子类只能继承自一个父类，Object 类是所有类的最终父类。

③ 多态性。多态性是指在一般类中定义的属性或行为。从同一类继承而来的不同子类，可以具有不同的数据类型或表现出不同的行为，即同样的消息被不同的对象接收时，可产生不同的行为。在同一个子类中，当给方法传递的参数不一样时也会产生不同的行为。

实现多态性的方法有覆盖和重载两种。覆盖是指子类重新定义父类的虚函数的做法；重载，是指允许存在多个同名函数，而这些函数的参数类型或数量不同。

例如，“动物”类有“叫”的行为。对象“猫”接收“叫”的消息时，叫的行为是“喵喵”，而对象“狗”接收“叫”的行为是“汪汪”。这就是用覆盖来实现这种多态性的。

再如，在窗体中画一个圆可调用以下类似方法。

```
SetCircular(5)
```

可以表示在窗体的正中心画一个 5 厘米的圆，此行为是由程序员定义的。该方法还可以有其他功能。例如，在窗体中画一个圆也可以调用以下类似方法。

```
SetCircular(20,30,5)
```

此方法与上面的方法名称是一样的，只是参数不一样，它表示在窗体的坐标(20,30)处画一个半径为 5 厘米的圆。这就是用重载来实现这种多态性的。

5.3 使用 Python 语言设计简单程序

程序设计是程序员利用计算机编程语言为解决某个问题而设计算法并编写程序过程，是软件工程过程中的重要组成部分。本节以 Python 语言为开发环境，通过介绍 Python 语言的数据结构、基本语法等知识，使读者学会应用使用 Python 语言解决简单问题，编写基本的基于 Python 的程序。