第5章

程序设计知识

要想让计算机完成某项工作，从简单的统计学生成绩到复杂的宇宙飞船自动控制系统，首先要明确给出工作步骤，然后用某种计算机能理解的方式告诉计算机。这就是计算机领域两项非常重要的工作：算法设计和程序设计，算法设计就是给出完成任务的工作步骤，程序设计就是用计算机能理解的某种语言把算法改写成程序。要想充分发挥计算机的作用，就必须针对要完成的工作，设计出高质量的算法和相应的程序，所以算法设计能力、程序设计能力是计算机专业学生必备的基本能力之一。

作为计算机专业的学生，首先是结合简单的问题（简单的算法），重点学习程序设计知识和提高程序设计能力。提高程序设计能力主要涉及三个方面的知识：一是语言知识，至少熟悉一种程序设计语言，能够根据算法思路熟练地编写出高质量的程序；二是数据结构知识，能够为要解决的问题设计出高效的数据逻辑结构和存储结构，保证程序的运行效率；三是编译知识，深入理解高级语言源程序的执行过程，有助于编写出高质量的中大规模程序。在具备了一定的程序设计知识和能力之后，再学习算法设计知识和提高算法设计能力。为此，本章对相关知识作简要介绍，详细内容在后续的高级语言程序设计、数据结构、编译原理、算法设计与分析等课程中介绍。

5.1 程序设计语言

程序设计能力是计算机专业人员与非专业人员的重要区别，虽然在现代程序开发环境的支持下，非计算机专业人员也能编写程序甚至比较复杂的程序，但总体来看，比较通用的功能强大的复杂程序，仍然是计算机专业人员或以计算机专业人员为主进行开发，程序设计知识的掌握和程序设计能力的提高是计算机专业学生胜任专业工作的重要能力之一。

如果要让助手帮助你完成某项工作，需要把工作步骤、注意事项等用书面语言或口头语言的方式告诉他。同样，要想让计算机完成某项工作，也需要把工作步骤等告诉计算机，而且要更明确、更详细。现在一般都是用书面语言方式，口头语言方式还在研究中。简单地说，适用于告诉计算机完成某项工作的语言就是程序设计语言。严格一点说，程序设计语言是一种让人与计算机之间进行交流，让计算机理解人的意图并按照人的意图完成工作的符号系统。针对要完成任务的步骤，基于某种程序设计语言编写出程序，提交给计算机执行，从而完成该项任务。

程序设计语言是指令或语句的集合，指令或语句是让计算机完成某项功能的命令，在机器语言或汇编语言中，把这样的命令称为指令（instruction），在高级语言中，把这样的命令称为语句（sentence）。程序设计语言经历了机器语言、汇编语言和高级语言三个阶段，机器语言和汇编语言都称为低级语言，高级语言分为结构化程序设计语言、面向对象程序设计语言、可视化程序设计语言、面向人工智能的程序设计语言和数据库语言等。

5.1.1 机器语言

1952年之前，人们只能使用机器语言来编写程序。机器语言（machine language）是由二进制编码指令构成的语言，是一种依附于机器硬件的语言。

每种处理器都有自己专用的机器指令集合，这些指令能够被计算机直接执行。由于指令的个数有限，所以处理器的设计者列出所有的指令，给每个指令指定一个二进制编号，用来表示这些指令。每条机器语言指令只能完成一个非常简单的任务，即使是求两个数的和这样的简单工作，也需要4条机器语言指令。

一条机器语言指令由两个部分组成：操作码（op-code）和操作数（operand），操作码用于说明指令的功能，操作数用于说明参与操作的数据或数据所在单元的地址。操作码和操作数都是以二进制的形式表示。

【例5.1】 机器语言程序示例。

程序功能：把两个内存单元中的数相加，并将结果存入另外一个单元。

部分程序如下：

```
0001 0101 01101100  //把地址为 01101100 的内存单元中的数装入 0101 号寄存器
0001 0110 01101101  //把地址为 01101101 的内存单元中的数装入 0110 号寄存器
0101 0000 01010110  //把 0101 和 0110 两个寄存器中的数相加，结果存入 0000 号寄存器
0011 0000 01101110  //把 0000 号寄存器中的数存入地址为 01101110 的内存单元中
```

用机器语言编写程序，程序员必须要记住每条指令对应的二进制数是什么，编写出来的程序就是由0和1组成的数字串。这样就存在几个方面的困难：指令难以准确记忆、程序容易写错、程序难以理解和修改。

如果有机会，真应该体验一下用机器语言编写程序的过程，这样就能体会到今天的高级语言为编写程序带来了多么大的方便。

5.1.2 汇编语言

1952年，出现了汇编语言。汇编语言（assembly language）是由助记符指令构成的语言，也是一种依附于机器硬件的语言。

在汇编语言中，使用助记符来表示指令的操作码（如用MOV表示数据传送操作，用ADD表示加法操作等），使用存储单元或寄存器的名字表示操作数。这样，相对于机器语言，记忆汇编语言的指令就容易多了，编写出的程序也比较容易理解。

【例5.2】 汇编语言源程序示例。

程序功能：把两个内存单元中的数据相加，并将结果存入另外一个单元。

部分程序如下：

```
MOV R5,X   //把 X 内存单元中的数装入 R5 寄存器
ADD R5,Y   //把 R5 中的数与 Y 单元中的数相加，结果再存回 R5 寄存器
MOV Z,R5   //把 R5 中的数存入 Z 单元中
```

和机器语言程序比较，实现的功能相同，但指令容易记忆，程序容易编写和理解，而且3条汇编语言指令完成了4条机器语言指令的功能。

实际上，计算机只能直接执行由机器语言编写的程序。用汇编语言编写的程序称为汇编语言源程序，需要首先翻译成功能上等价的机器语言程序（称为目标程序），才能被计算机执行，完成这种翻译工作的程序称为汇编程序或汇编器（assembler）。

相对于机器语言，汇编语言有一定的优势，但仍存在许多不足，助记符对一般人来说仍是比较难以记忆的，而且需要编程人员对计算机的硬件结构有比较深入的了解。

5.1.3　高级语言

机器语言中的指令用二进制数字串表示，汇编语言中的指令用英文助记符表示，高级语言（high level language）中的语句用英文和数学公式表示，更容易被编程人员理解和掌握。

【例 5.3】 高级语言源程序示例。

程序功能：把两个内存单元中的数相加，并将结果存入另外一个单元。

部分程序如下：

```
Z = X + Y    //把内存单元 X 中的数与 Y 中的数相加，结果存入 Z 单元
```

从这个简单的例子可以看出，用高级语言编写程序，既简单又容易理解。

使用高级语言编写出的程序称为高级语言源程序，也需要先翻译成等价的目标程序，才能为计算机理解和执行。这种翻译程序有两种模式，一种是编译程序模式，一种是解释程序模式。编译程序先把高级语言的源程序翻译成目标程序，然后执行目标程序；解释程序并不需要把高级语言的源程序翻译成目标程序，而是边翻译边执行。

由于机器语言实在是难以学习和理解，所以一般不直接用机器语言编写程序。相对于高级语言，汇编语言也有难以学习和理解的不足，但汇编语言靠近机器，能够充分利用计算机硬件的特性，所以编写出的程序效率较高（占用内存少、执行速度快），对效率要求较高的规模不大的程序（如外设驱动程序、计算机控制程序等）仍然可用汇编语言编写。但更多的规模比较大的程序还是用高级语言来编写。

第一个实用高级语言是美国 IBM 公司的约翰·巴克斯（John W. Backus）等人于 1957 年研制成功的 FORTRAN。几十年来，人们提出了几千种方案，能实现的也有几十种。我们只对得到广泛应用的主要高级语言作简要介绍。

1. FORTRAN 语言

公式翻译器（formula translator，FORTRAN），用于数学公式的表达和科学计算，1957 年 FORTRAN 的第一个版本研发成功，其编译程序由 25 000 行机器语言指令组成，耗资 250 万美元，安装在 IBM 704 计算机上使用。以后又陆续研发出 FORTRAN 66、FORTRAN 77、FORTRAN 90、FORTRAN 95 和 FORTRAN 2003 等版本，FORTRAN 77 是一种良好的结构化程序设计语言，FORTRAN 2003 是面向对象的程序设计语言。

目前，作为优秀的科学计算语言，FORTRAN 在计算密集的分子生物学、高能物理学、大气物理学、地质学和气象学（天气预报）等领域仍然得到广泛的应用。

2. ALGOL 语言

算法语言（algorithm language，ALGOL），也是用于科学计算，其最早版本是 1958 年出现的 ALGOL 58，后续版本有 ALGOL 60 和 ALGOL 68，这两个版本曾经在我国得到广泛的学习和使用，其后继语言 Pascal 出现后，ALGOL 逐渐被淘汰。

3. COBOL 语言

面向商业的通用语言(common business-oriented language,COBOL),用于企业管理和事务处理,以一种接近于英语书面语言的形式来描述数据特性和数据处理过程,因而便于理解和学习。

20 世纪 50 年代中期,计算机开始用于商业和企业的事务处理,而事务处理与科学计算不同,数据繁多而运算简单,它只需要一定的运算能力,但对数据结构的描述和大批量数据的分析处理方面则要求有很强的能力。1959 年 5 月美国国防部召开专门会议,讨论研发通用商业语言的要求和可能性,确定了这种语言的基本设计思想和应具有的特点。1960 年 4 月正式公布第一个 COBOL 文本,称为 COBOL 60,后续版本有 COBOL 65、COBOL 68、COBOL 72、COBOL 78、COBOL 85 和 COBOL 2002 等。

现在,在银行等行业仍有 COBOL 程序在运行,在大中型机环境下,COBOL 仍是一种可选用的程序设计语言。

4. BASIC 语言

初学者通用符号指令码(beginner's all-purpose symbolic instruction code,BASIC)。BASIC 的研发者认为,FORTRAN、ALGOL 和 COBOL 语言都是面向计算机专业人员的,为使各专业的大学生都能较快地掌握一种编程语言,研发了 BASIC 语言。

1964 年的 BASIC 第 1 版只有 14 条语句,到 1971 年的第 6 版已完善成为相当稳定的通用语言。以后陆续推出了各种版本的 BASIC,主要有 Apple BASIC、MS BASIC(BASICA)、GWBASIC、True BASIC、Quick BASIC、Turbo BASIC 和 Visual Basic 等,其中 True BASIC、Quick BASIC 和 Turbo BASIC 是结构化程序设计语言,Visual Basic 是面向对象的程序设计语言。

BASIC 确实简单、易学,一经推出,很快流行起来,几乎所有小型、微型计算机,甚至部分大中型计算机,都配有 BASIC 语言。BASIC 语言在我国也得到广泛流行。谭浩强教授编写的《BASIC 语言》一书,销售量超过 1000 万册,从一个侧面说明了 BASIC 语言在当时的流行程度。

5.1.4 结构化程序设计语言

20 世纪 50—60 年代,由于计算机硬件环境(运算速度慢、内存容量小)、编程语言、应用领域等的限制,编写的程序一般都比较小,编程人员更多的是注重程序功能的实现和编程技巧,在实现功能的前提下,尽可能少占用内存空间并具有较高的运行效率。

到了 20 世纪 60 年代末,随着计算机硬件水平的提高和应用的深入,需要编写规模较大的程序,如操作系统、数据库管理系统等。实践表明,沿用过去编写小程序的方法(注重功能的实现,注重内存的节省,注重程序执行效率的提高;不注重程序结构的清晰性,不注重程序的可理解性和可修改性)编写中大规模的程序是不行的,往往导致编写出的程序可靠性差、错误多且难以发现和修改错误。为此,人们开始重新审视程序设计中的一些基本问题,如程序的基本组成部分是什么、如何保证程序的正确性、程序设计方法如何规范等。

1969 年,埃德斯加·狄克斯特拉提出了结构化程序设计(structured programming,SP)的概念,强调从程序结构和风格上来研究程序设计,注重程序结构的清晰性,注重程序的可理解性和可修改性。对于编写规模比较大的程序,不可能不犯错误,关键的问题是在编写程序时就应该考虑到,如何较快地找到程序中的错误并较容易地改正错误。经过几年的探索和实践,结

构化程序设计方法的应用确实取得了成效，遵循结构化程序设计方法编写出来的程序，不仅结构良好，容易理解和阅读，而且容易发现和改正错误。

到 20 世纪 70 年代末，结构化程序设计方法得到了很大的发展，尼克莱斯·沃思(Niklaus Wirth)提出了“算法＋数据结构＝程序设计”的程序设计方法，将整个程序划分成若干个可单独命名和编址的部分——模块。模块化实际上是把一个复杂的大程序的编写分解为若干个相互联系又相对独立的小程序的编写，使程序易于编写、理解和修改。在 20 世纪 80 年代，模块化程序设计方法广泛流行。

好的程序设计方法要有相应的程序设计语言支持，1971 年，尼克莱斯·沃思研发了第一个结构化程序设计语言 Pascal，后来出现的 C 语言也属于结构化程序设计语言，从前面的介绍可知，FORTRAN、COBOL 和 BASIC 也都有结构化版本。

1. Pascal 语言

Pascal 是一种通用的高级语言，是在 ALGOL 语言的基础上发展起来的，以法国著名科学家帕斯卡(Blaise Pascal)的名字命名，这位物理学家、数学家在 1642 年曾经发明了齿轮式、能进行加减运算的机械式计算机，著名的帕斯卡定律就是他发现的。

Pascal 语言的主要特点是严格的结构化形式，丰富完备的数据类型，运行效率高，查错能力强。Pascal 语言对于培养初学者良好的程序设计风格和习惯很有益处。

Pascal 的第一个版本出现在 1971 年，之后出现了适合于不同机型的各种版本。其中影响最大的就是 Turbo Pascal 系列，它是由美国 Borland 公司研发的一种适用于微型计算机的 Pascal 语言，从 1983 年推出 Turbo Pascal 1.0，一直到 1992 年推出的 Turbo Pascal 7.0，其功能不断完善。从 1989 年的 Turbo Pascal 5.5 开始支持面向对象的程序设计。20 世纪 70—90 年代，Pascal(Turbo Pascal)语言有很大的影响，现在人们很少再学习、使用 Pascal 了，但人们目前使用的数据库应用系统开发工具 Delphi 就是在 Pascal 的基础上发展起来的。

2. C 语言

C 语言的前身是 ALGOL 60。ALGOL 60 是一种面向问题的高级语言，它描述算法很方便，但是它离硬件比较远，不适合用来编写系统软件(如操作系统)。1963 年，英国剑桥大学在 ALGOL 60 的基础上添加了硬件处理功能，推出了 CPL(combined programming language)。但 CPL 规模比较大，难以实现。1967 年剑桥大学的 Matin Richards 对 CPL 语言作了简化，推出了 BCPL(basic combined programming language)。1970 年美国贝尔实验室以 BCPL 语言为基础，又作了进一步简化，设计出更简单且更接近硬件的 B(取 BCPL 的第一个字母)语言，并用 B 语言编写了第一个高级语言版的 UNIX 操作系统。但 B 语言过于简单，功能有限。1972—1973 年，贝尔实验室在 B 语言的基础上设计出了 C(取 BCPL 的第二个字母)语言。C 语言既保持了 BCPL 和 B 语言精练、接近硬件的优点，又克服了它们过于简单、无数据类型的缺点。

1973 年，贝尔实验室将 1969 年用汇编语言编写的 UNIX 操作系统用 C 语言改写成 UNIX 第 5 版，C 语言代码占 90％以上，只对最关键部分保留汇编语言代码，这样就使得 UNIX 操作系统向其他机器移植变得简单。1975 年，UNIX 第 6 版公布后，C 语言的优点引起人们的普遍注意，随着 UNIX 的日益广泛使用，C 语言也迅速得以推广，1978 年以后，C 语言广泛应用到大、中、小和微型计算机上。

最初，C 语言是为编写 UNIX 操作系统而研发的，但由于 C 语言的强大功能和各方面的优点逐渐为人们认识，C 语言得以迅速传播，成为当代最优秀的程序设计语言之一。目前，在

微型计算机上得到广泛应用的是 Turbo C、MS C、Quick C、C++、C#、Visual C++和 Visual C++.NET 等版本。其中，后 4 个版本支持面向对象的程序设计。

5.1.5 面向对象程序设计语言

几十年的程序设计实践表明，结构化程序设计方法在一定程度上保证了编写较大规模程序的质量，但随着时间的推移，也逐渐暴露了其本身存在的不足。

（1）面向过程的设计方法与人们习惯的思维方式仍然存在一定的距离，所以很难自然、准确地反映真实世界，因而用此方法编写出来的程序，特别是规模比较大的程序，其质量仍然是难以保证的。

（2）结构化程序设计方法强调了要实现功能的操作方法（模块），而被操作的数据（变量）处于实现功能的从属地位，即程序模块和数据结构是松散地耦合在一起，当程序复杂度较高时，容易出错，而且错误难以查找和修改。

为了弥补结构化程序设计方法的不足，适应程序设计的需要，20 世纪 80 年代，在程序设计中各种概念和方法积累的基础上，就如何超越程序的复杂性障碍，如何在计算机系统中自然地表示客观世界等问题，人们提出了面向对象的程序设计（object oriented programming，OOP）方法。

面向对象的方法不再将问题分解为过程，而是将问题分解为对象，对象将自己的属性和方法封装成一个整体，供程序设计者使用，对象之间的相互作用则通过消息传递来实现。使用面向对象的程序设计方法，可以使人们对复杂系统的认识过程与程序设计过程尽可能一致。这种“对象＋消息”的面向对象程序设计方法正逐渐取代“数据结构＋算法”的面向过程的程序设计方法（结构化程序设计方法）。

像结构化程序设计方法要有结构化程序设计语言支持一样，面向对象的程序设计方法也要有面向对象的程序设计语言支持。

1. Simula 67

Simula 67 发布于 1967 年，被公认为是面向对象语言的鼻祖。Simula 67 的基础是 ALGOL 60，Simula 67 具有类和对象的概念，20 世纪 80 年代美国 Xerox Palo Alto 研究中心推出了 Smalltalk，它完整地体现并进一步丰富了面向对象的概念，开发了配套的工具环境。但由于当时人们已经接受并广泛应用结构化程序设计方法，一时还难以完全接受面向对象的程序设计思想，这类纯面向对象语言没有能够广泛流行起来。后来，人们对已经流行的语言进行面向对象的扩充，曾经推出过许多种版本，成功的代表是在流行的 C 语言基础上开发的 C++语言。

2. C++

C++语言最先由 AT&T 公司贝尔实验室计算机科学研究中心的 Bjarne Stroustrup 在 20 世纪 80 年代初设计并实现，它是以 C 语言为基础的支持数据抽象和面向对象风范的通用程序设计语言。C++是 C 语言的扩充，从 Simula 67、ALGOL 68 和 Ada 等语言中吸取了多种先进特性，并保持了 C 语言紧凑、灵活、高效和移植性好的优点，比 Smalltalk 等面向对象语言具有更好的性能，再加上 C 语言的普及基础，C++是得到广泛应用的一种面向对象语言，在得到广泛应用的同时仍在不断发展和改进。

C++支持数据的封装，支持类的继承，也支持函数的多态，这都提高了程序的可扩展性和可重用性，进而提高了软件开发的效率。例如，编写一个计算不同几何图形面积的程序，可以

将几何图形的形状定义为一个基类，由它派生出一些子类，如圆形、矩形等，它们具有基类的共性，又有各自的特性。用动态联编来实现运行时的多态，使得在不同类中对相同名字的函数进行选择，实现不同图形面积的计算，动态联编通过虚函数来实现，它在各个子类中都有不同的实现。而要在 C 语言中实现该功能，需要编写不同的子函数，还要通过函数调用来实现不同图形面积的计算，其中公共部分需要多次定义，代码冗余。另外，如果想要再增加新的几何图形面积计算，对于 C++来说也是比较方便的，只要再定义图形基类的一个新的子类，并在该子类中给出求几何图形面积的方法即可，而对于 C 语言，需要重新定义一个子函数。

3. Java

Java 语言是由 Sun Microsystems 公司于 1995 年 5 月推出的一个支持网络计算的面向对象程序设计语言。Java 语言吸收了 Smalltalk 语言和 C++语言的优点，并增加了并发程序设计、网络通信和多媒体数据控制等特性，也是目前得到广泛应用的一种面向对象程序设计语言。

4. C#

C#语言是微软公司发布的一种面向对象的、运行于. NET Framework 之上的高级程序设计语言。C#在语法规则与系统结构上与 Java 有着很多的相似之处，比如它包括了单继承机制、界面以及与 Java 几乎相同的语法，是微软公司基于. NET 网络框架进行系统开发的主角。

5. Python

Python 是一种完全面向对象的、解释型的程序设计语言，它由荷兰的 Guido van Rossum 于 1989 年开发完成，并于 1991 年发行了第一个公开版本，目前的最新版本是 Python 3. 7. 3。由于 Python 语言的简洁、易读、易维护以及有大量的内置库和第三方库可用，使得它成为一种广受欢迎、得到广泛应用的程序设计语言。

5.1.6　可视化程序设计语言

近些年，程序设计的观念发生了显著变化，可视化(visual)技术广泛用于各种程序设计过程，基于 C++，就有 C++Builder 和 Visual C++可视化程序设计语言。这些可视化语言以其图形化的编程方式将面向对象技术的特性体现出来，通过用鼠标拖曳图形化的控件就可以完成 Windows 风格界面的设计工作，Windows 风格界面主要由窗口、按钮、菜单等元素组成，大大减轻了程序设计人员的编程工作量，使得开发软件这一原本枯燥、难以理解的工作变得相对轻松快捷。在 2002 年初，微软公司又推出了 Visual C++的最新版本——Visual C++. NET，它继承了以往 Visual C++各版本的优点，增加了许多新的特性，使得开发能力更强、开发的效率更高。Visual Basic 继承了 BASIC 简单易学的特点，也是得到广泛应用的可视化程序设计语言，特别适合于初学者和非专业人员。

5.1.7　人工智能程序设计语言

人工智能是让计算机具有类似于人的智能，完成诸如判断、推理、证明、识别、学习等智能性工作。实际上，计算机所做的所有工作都是在程序的支持下完成的，同样程序设计也是实现人工智能的关键。人工智能程序要能有效地处理知识表示和逻辑推理，擅长数值计算和事务处理的 FORTRAN、COBOL、BASIC、Pascal 和 C 语言等不大适合于编写人工智能程序。人们开发出了适合于知识表示和逻辑推理的人工智能语言，主要有 LISP 和 PROLOG。

1. LISP 语言

LISP 是表处理(LISt Processing)的缩写，LISP 语言在 1958 年由美国麻省理工学院的人工智能小组提出，1960 年由约翰·麦卡锡(John McCarthy)教授整理成统称为 LISP 1.0 的形式发表，以后陆续出现了 LISP 1.5、LISP 1.6、MACLISP、INTERLISP、COMMONLISP、GCLISP 和 CCLISP 等版本，其中 INTERLISP、MACLISP 和 COMMON LISP 最为流行，得到广泛应用。在 LISP 语言中设计了一套符号处理函数，它们具有符号集上的递归函数的计算能力，原则上可以解决人工智能中的任何符号处理问题。LISP 语言的主要不足，一是数据类型少，表达能力有限；二是程序执行速度较慢。

2. PROLOG 语言

PROLOG 是逻辑程序设计(PROgramming in LOGic)的缩写，1972 年法国马赛大学的 Alain Colmerauer 等人设计实现了解释性 PROLOG 语言，在欧洲人工智能领域得到广泛应用。

PROLOG 基于一阶谓词逻辑，既有坚实的理论基础，又有较强的表达能力。PROLOG 语言自动实现模式匹配、回溯这两种人工智能中常用的基本操作。其主要不足是系统开销较大、程序执行效率较低。

在人工智能领域中，LISP 语言和 PGOLOG 语言仍在使用，最近几年，又出现了可视化的 Visual LISP 和 Visual PGOLOG。

计算机的一个重要应用领域是数据处理，其基础是数据库，编写程序用的是数据库语言，具体介绍见 6.1 节。

5.2 Python 语言程序设计

5.2.1 Python 语言的特点

相对于其他程序设计语言(如 C、C++、Java 等)，Python 语言主要有两个方面的特点，一是易学易用，二是有丰富的第三方库可用。这两个特点，使得 Python 语言得到了广泛的学习和使用。

1. 易学易用

Python 语言的语法很多来自于 C 语言，但比 C 语言更为简洁。相对于其他常用程序设计语言，可以用更少的代码实现相同的功能，也更容易学习掌握和使用，这可使编程人员更多地关注数据处理逻辑，而不是语法细节。

2. 类库丰富

Python 解释器提供了几百个内置类库，此外，世界各地的程序员通过开源社区贡献了十几万个第三方库，几乎覆盖了计算机技术的各个领域，编写 Python 程序可以大量利用已有的内置类库和第三方库中的函数。在一定程度上说，使用 Python 语言编写程序，是基于大量的现成函数(代码)来组装程序，大大减少了编程人员自己编写代码的工作量，简化了编程工作，提高了编程效率，提升了代码质量。随着计算机硬件性能的不断提高，在一般的应用场合，Python 程序的性能表现与 C++、Java 等语言已没有明显的区别。

5.2.2 Python 的安装

学习编程首先要搭建一个编程环境，搭建 Python 编程环境主要是安装

Python 解释器。有了 Python 解释器的支持，才能执行 Python 程序和语句，才能验证编写的程序是否正确以及执行效率如何。

Python 解释器可以在 Python 语言官网下载后安装。目前的最新版本是 Python 3.7.0。以 Python 3.6.6 的安装为例，Python 的安装过程一般包含如下 3 个主要步骤：

(1) 下载安装包。安装 Python，首先需要做的就是访问 Python 官方网站：http://www.python.org/download/，从官方网站下载 Python 的安装包。访问 Python 官网进入如图 5.1 所示的 Python 主界面。

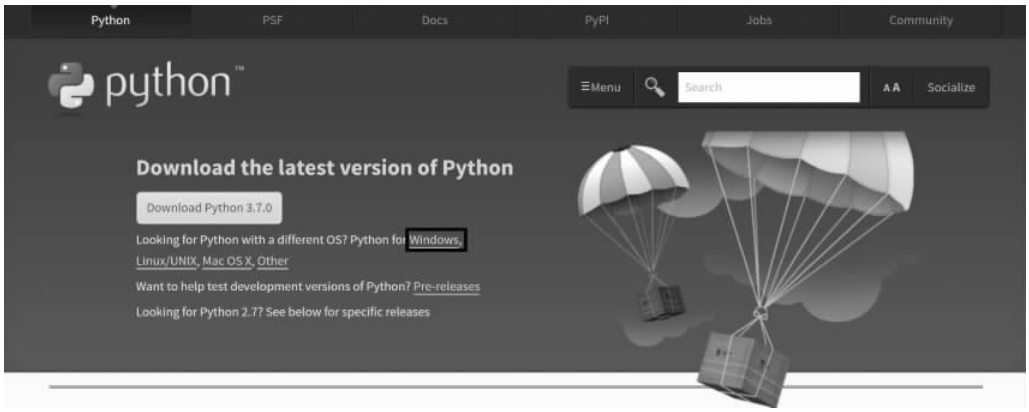

图 5.1　Python 官网主界面

如果所用计算机安装的是 Windows 操作系统，应选择单击 Python for Windows(也可以根据所用操作系统有不同的选择)，进入面向 Windows 的 Python 版本列表，从中找到 Python 3.6.6-2018 下的 Download Windows x86-64 executable installer 选项并单击，下载安装包文件，如图 5.2 所示。

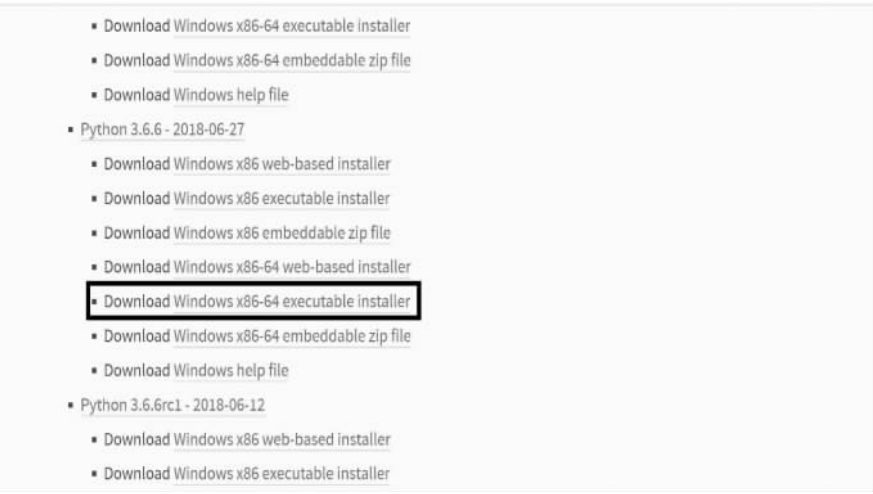

图 5.2　Python 3.6.6 版本选项

(2) 安装 Python 解释器。双击下载的 Python 安装包文件，单击 Install Now 选项后进入安装过程，如图 5.3 所示。为了后续操作方便，请选中 Add Python 3.6 to PATH 复选框。

图 5.3 Python 安装起始对话框

（3）安装成功。安装过程结束后，出现如图 5.4 所示的界面，单击 Disable path length limit 方框取消路径长度限制后，单击 Close 按钮完成 Python 安装。也可以不取消路径长度限制，直接单击 Close 按钮完成 Python 安装。

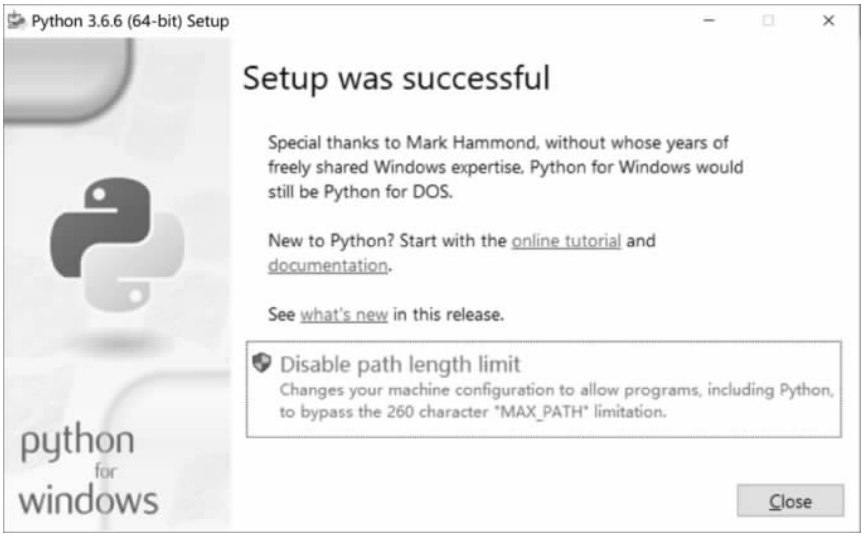

图 5.4 Python 安装结束对话框

安装完成后，可以从 Windows 开始菜单的所有程序（应用）中找到 Python 3.6.6 程序组，其中的 IDLE 即为 Python GUI（Python 图形用户界面），这就是 Python 的集成开发学习环境（Integrated Development and Learning Environment，IDLE）。其中“>>>”为 IDLE 的操作提示符，在其后面可输入并执行 Python 表达式或语句，输入表达式：25 * 36 并按回车键，则会显示计算结果：900；输入语句：print("Python Language")并按回车键，则会显示字符串：Python Language，如图 5.5 所示。

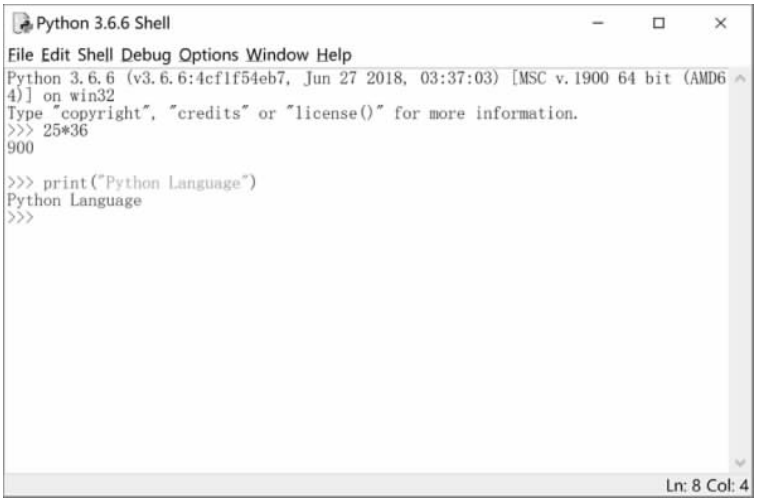

图 5.5　Python IDLE 界面

5.2.3　Python 程序的运行

安装好 Python 解释器后，运行 Python 程序有两种方式：命令行方式和程序文件方式。命令行方式也是一种人机交互方式，用户输入一行命令（一条 Python 语句），计算机就执行一条，并即时输出执行结果；程序文件方式是一种批量执行语句的方式，用户把若干条 Python 语句写入一个或多个程序文件中，然后执行程序文件。命令行方式用于验证、调试少量的语句代码，程序文件方式是更常用的方式，用于调试执行、修改完善由多条语句组成的程序。

1. 命令行方式

IDLE 实际上是一个 Python 的外壳（shell），它提供了交互式命令行的运行方式，可以在提示符（>>>）后面输入想要执行的语句，回车后可以立即显示运行结果。例如输入：

```
>>> print("Python Learning Environment.")
```

其中的 print()是 Python 提供的输出函数，其功能是显示输出表达式的计算结果，这条语句的功能是在屏幕上显示输出字符串："Python Learning Environment."。

这种单行命令的交互方式简单方便，可用于实现简单的功能和学习时的语法格式验证，但不适合解决复杂问题，只有编写程序才能更好地体现出 Python 作为编程语言的优势。

2. 程序文件方式

一般情形下，应该是把多条语句组织成一个程序文件，然后再执行程序文件，达到解决实际问题的目的。在如图 5.5 所示的 IDLE 界面中选择菜单 File→New File，打开程序编辑窗口，可以输入若干条语句，形成一个 Python 程序，如图 5.6 所示。选择菜单 File→Save，给定文件名并指定存储位置后，将程序存盘，Python 源程序文件的默认扩展名是.py。接下来可以选择 Run→Run Module 菜单项（或者按 F5 快捷键）运行程序，将在一个标记为 Python Shell 的窗口中显示运行结果。也可以在 IDLE 中选择菜单 File→Open…，打开一个已经存在的 Python 程序文件，用于编辑修改和调试执行。

图 5.6 Python 程序编辑窗口

5.2.4 Python 的基础语法

1. Python 标识符

标识符由字符集中的字符按照一定的规则构成。Python 中变量、函数、文件等各种实体的名字都需要用标识符来表示。Python 规定：标识符是由字母、数字和下画线 3 种字符构成的且第一个字符必须是字母或下画线的字符序列。

定义标识符时要注意如下几点：

(1) 必须以字母或下画线作为开始符号，数字不能作为开始符号，但以下画线开始的标识符一般都有特定含义，所以尽量不用以下画线开始的标识符。

(2) 标识符中只能出现字母、数字和下画线，不能出现其他符号。

(3) 同一字母的大写和小写被认为是两个不同的字符。

(4) 保留字(关键字)有特定的含义，不能用作用户自定义标识符使用。

(5) 尽可能做到见名知义，增加程序的可理解性。

2. 常量与变量

1) 常量

常量是指在整个程序的执行过程中其值不能被改变的量，也就是所说的常数。在用 Python 语言编写程序时，常量不需要类型说明就可以直接使用，常量的类型是由常量值本身决定的。如 12 是整型常量、34.56 是浮点型常量、'a'和"Python"是字符串常量等。

在 Python 中，常量主要包括两大类：数值型常量和字符型常量。数值型常量，简称数值常量，常用的数值型常量为整型常量和浮点型常量，即整数和实数。字符型常量就是字符串。

2) 变量

变量是指在程序运行过程中，其值可以被改变的量。变量要先定义(赋值)，后使用。

变量定义(赋值)格式如下：

变量名 1,变量名 2,…,变量名 n = 表达式 1,表达式 2,…,表达式 n

功能：为各变量在内存中分配相应的内存单元并赋以相应的值。分别计算出各表达式的

值，并依次赋给左边的变量。各变量名分别是一个合法的自定义标识符。在程序中变量用来存放初始值、中间结果或最终结果。

下面定义了 3 个整型变量和 2 个浮点型变量：

```
>>> i = 10                                  # 定义 1 个整型变量
>>> num,sum = 1,0                           # 定义 2 个整型变量
>>> length,height = 23.6,12.8               # 定义 2 个浮点型变量
```

变量的类型由其所赋值决定，随着赋值的改变，其类型也作相应改变。

示例：

```
>>> x = 10                                  # 为变量 x 赋以整型值，变量的类型为整型
>>> x = 12.5                                # 为变量 x 赋以浮点值，变量的类型变为浮点型
```

说明：

(1) 由井字号(#)开始至行末的内容为注释，用于说明语句或程序的功能，便于人们阅读理解程序或语句，不影响程序的执行。

(2) 变量是自定义标识符的一种，当然要遵守标识符的命名规则。除此之外，为增加程序的可读性，一般约定变量名全部用小写字母，多个单词之间用下画线连接或者将非第一个单词的第一个字母大写。本书中，变量名选用多个单词之间用下画线连接的方式，如 total_weight。

5.2.5　Python 的基本数据类型

程序的功能是处理数据，不同类型的数据有不同的存储方式和处理规则。所以，在程序中首先要明确待处理数据的类型，才能使数据得以正确存储和处理。

Python 中提供了多种数据类型，包括整型、浮点型、布尔型和字符串型等基本数据类型和列表、字典等组合数据类型。此处先介绍几种基本数据类型，组合数据类型的介绍在 5.3 节结合数据结构进行。

1. 整型

整型就是整数类型，如 365、－126、78、220 等都是整型数据。

Python 中整数的取值范围很大，理论上没有限制，实际取值受限于所用计算机的内存容量，对于一般的计算应该足够用了。

示例：

```
>>> 12345678987654321 * 12345678987654321
152415789666209420210333789971041     # 运算结果
```

2. 浮点型

浮点型就是实数类型，表示带有小数的数值(由于小数点的位置是浮动的，也称为浮点数)。Python 语言要求所有浮点数都必须带有小数，便于和整数区别，如 6 是整数，6.0 是浮点数。虽然 6 和 6.0 值相同，但两者在计算机内部的存储方式和计算处理方式是不一样的。

浮点数有两种表示方式：小数方式和科学记数方式。3.14、1.44、－1.732、19.98、9000.0 等都是浮点数的小数表示方式；31.4e-1、0.0314e2、－173.2E-2、9.0E3 等都是浮点数的科学记数表示方式，科学记数表示方式使用字母 e 或 E 代表以 10 为基数的幂运算，31.4e-1 表示 31.4×10^{-1}。

示例：

```
>>> 3 + 2                          # 两个整数相加
5                                  # 结果为整数
>>> 3.0 + 2                        # 浮点数加整数
5.0                                # 结果为浮点数
>>> 3.1e2 + 3.2e5                  # 浮点数加浮点数
320310.0                           # 结果为浮点数
```

3. 布尔型

布尔型也称为逻辑型，用于表示逻辑数据。Python 中，逻辑数据只有两个值：False（假）和 True（真）。需要注意的是，两个逻辑值的首字母大写，其他字母小写。FALSE，false，TRUE，true 等书写方式都不是正确的 Python 逻辑值。

示例：

```
>>> a = True                       # 给变量赋予逻辑值 True
>>> b = False                      # 给变量赋予逻辑值 False
>>> print(a,b)                     # 输出逻辑变量的值
True False
```

4. 字符串型

1）字符串定义

字符串型数据用于表示字符序列，字符串常量是由一对引号括起来的字符序列。Python 中有 3 种形式的字符串：

一对单引号括起来的字符序列，如'Python'、'程序设计'；

一对双引号括起来的字符序列，如"Python"、"程序设计"；

一对三引号括起来的字符序列，如'''Python'''、'''程序设计'''。

几点说明如下：

（1）单引号或双引号括起来的字符串只能书写在一行内，三引号括起来的字符串可以书写多行。

（2）单引号括起来的字符串中可以出现双引号，双引号括起来的字符串中可以出现单引号，三引号括起来的字符串中可以出现单引号和双引号。

示例：

```
>>> print('''学习'程序设计'是培养"计算思维"的有效方式''')
学习'程序设计'是培养"计算思维"的有效方式
```

（3）字符串有两个特例，一个是单字符字符串（可称为字符），另一个是不包含任何字符的字符串（称为空字符串）。

示例：

```
str1 = 'A'                         # 只包含单个字符 A 的字符串
str3 = " "                         # 只包含单个空格的字符串
str4 = ""                          # 不包含任何字符的字符串
```

空格字符串和空字符串是不同的字符串，前者包含一个或多个空格，字符串长度不为 0，后者不包含任何字符，字符串长度为 0。

（4）由三引号括起来的字符序列，如果出现在赋值语句中或 print()函数内，当作字符串处理；如果直接出现在程序中，当作程序注释。

2）转义字符

Python 中，以 Unicode 编码存储字符串，字符串中的单个英文字符和中文字符都看作 1 个字符。Unicode 编码表中，除了一般的中英文字符外，还有多个控制字符，要是用到这些控制符，只能写成编码值的形式。如 10 表示换行、13 表示回车等。

直接书写编码值的方式是比较麻烦的，也容易出错。为此，Python 给出了一种转义符的表示形式，以反斜杠(\)开始的符号不再是原来的意义，而是转换为新的含义。如\n 代表换行符，\r 代表回车符，\t 代表水平制表符等。

示例：

```
>>> print("Python\n 程序设计")          # 换行后输出"程序设计"
>>> print("Python\t 程序设计")          # 在下一个制表符位置输出"程序设计"
```

3）字符串的访问

对于字符串，除了可以整体使用外，还有两种常用的访问方式：索引方式和切片方式。

索引访问方式也称为单字符访问方式，语法格式如下：

字符串变量名[索引值]

功能：从字符串中取出与索引值对应的一个字符。字符串中每个字符都对应一个索引值，有两种索引值设置方式：正向递增方式(从 0 开始)和逆向递减方式(从－1 开始)。

示例：

```
>>> str1 = "ABC 计算机"                # 对应的索引值如图 5.7 所示
>>> ch1 = str1[2]                      # 值为"C"
>>> ch2 = str1[4]                      # 值为"算"
>>> ch3 = str1[ - 1]                   # 值为"机"
```

图 5.7　字符串“ABC 计算机”对应的索引值

说明：

(1) 正向索引的开始值为 0(不是 1)，逆向索引的开始值为－1。

(2) 对于单个符号，不管是英文字符、数字字符，还是汉字(也可称为汉字字符)，都按一个字符对应索引值。

切片访问方式也称为子串访问方式，语法格式如下：

字符串变量名[i:j:k]

功能：从字符串中取出多个字符。其中，i 为开始位置，j 为结束位置(但取出的字符中不包括 j 位置上的字符，是截止到 j－1 位置上的字符)，k 为步长。参数 i、j、k 都可以省略。在步长 k 的值为正数时，省略 i，其默认值为 0；省略 j，其默认值为正向最后一个字符的索引值加 1。在步长 k 的值为负数时，省略 i，其默认值为－1；省略 j，其默认值为逆向最后一个字符的索引值减 1。省略 k，其默认值为 1。省略 k 时，其前面的冒号可以省略(当然也可以不省略)，

省略i或j(或i和j都省略)时,二者之间的冒号不能省略。

示例：

```
>>> str1 = "ABC计算机"
>>> str1[3:5]                    # 值为'计算'
>>> str1[3:-1:2]                 # 值为'计'
>>> str1[:3]                     # 值为'ABC'
>>> str1[3:]                     # 值为'计算机'
```

4）字符串运算符

Python中可以进行字符串的连接、比较以及判断子串等运算,运算符及功能如表5.1所示。

表5.1 字符串运算符

运 算 符	示例与功能描述
+	str1+str2：连接字符串str1和str2
*	str1*n或n*str1：字符串str1自身连接n次
in	str1 in str2：如果str1是str2的子串,返回True,否则返回False
not in	str1 not in str2：如果str1不是str2的子串,返回True,否则返回False
<,<=,>,>=,==,!=	str1<str2：如果str1小于str2,返回True,否则返回False str1==str2：如果str1和str2相等,返回True,否则返回False 其他比较运算类似

注：str1和str2可以是字符串变量名或字符串常量。

5）字符串运算函数

常用的4个字符串运算函数如表5.2所示。

表5.2 常用的字符串运算函数

函 数 名	示例与功能描述
len(字符串)	len(str1)：返回字符串str1的长度,即字符串中字符的个数
str(数值)	str(x)：返回数值x对应的字符串,可以带正负号
chr(编码值)	chr(n)：返回整数n对应的字符,n是一个编码值
ord(字符)	ord(c)：返回字符c对应的编码值

注：表中的编码值是指Unicode编码值。

示例：

```
>>> len("Python程序设计")          # 结果为10,英文字母和汉字都按1个字符计算
>>> str(-67.5)                    # 结果为字符串"-67.5"
>>> chr(65)                       # 结果为字符"A"
>>> ord("A")                      # 结果为整数65
```

5.2.6 Python的类型转换

编写程序时,经常会遇到不同类型数据之间的混合运算。不同类型数据之间的运算是可以的,但由于不同类型数据的存储格式是不一样的,所以要先进行相应的类型转换之后,才能进行运算。这种类型转换有2种方式：一是自动类型转换,也称隐式类型转换,不需要编程人员书写相关要求,由Python解释器自动进行；二是强制类型转换,也称显式类型转换,由编程

人员在程序中书写出类型转换要求。

Python 的自动类型转换规则为：算术表达式中的类型转换以保证数据的精度为准则，即整数与浮点数进行混合运算时，要把整数转换为浮点数。

如果自动类型转换不符合特定计算的需要，可由编程人员在编写程序时强行把某种类型转换为另一种指定的类型，称为强制类型转换。

强制类型转换通过如下两个函数实现：

```
int(x)
float(x)
```

int(x)函数的功能是把 x 的值转换为整型数据，x 为浮点数或由数字组成的字符串。float(x)函数的功能是把 x 的值转换为浮点型数据，x 为整数或由数字与最多一个小数点组成的字符串。

对于类型转换，还有一个功能更强大的 eval()函数。把由纯数字组成的字符串(可由正负号开始)转换为整型，把由数字和 1 位小数点组合成的字符串(可由正负号开始，可以是指数表示形式)转化为浮点型数据，还可以使用 eval()函数。

从类型转换的角度看，一个 eval()函数的功能相当于 int()和 float()两个函数的功能。不仅如此，eval()函数还有更多、更灵活的功能。

eval()函数的语法格式如下：

```
eval(字符串)
```

功能：将字符串的内容(去掉引号)看作一个 Python 表达式，并计算出表达式的值作为函数的结果。字符串以单引号、双引号、三引号形式书写都可以。

示例：

```
>>> eval('3.14 * 5 * 5')                    # 单引号字符串
78.5
>>> a,b = 3,5
>>> eval("a * 6 + b")                       # 带变量的表达式,变量要先定义
23
```

eval()函数给表达式的计算带来了方便，如下的语句相当于一个功能强大的计算器，可以计算出用户输入的算术表达式的结果值：

```
>>> print(eval(input("表达式 = ")))
表达式 = 3.14 * 5 * 5                       # 输入表达式 3.14 * 5 * 5
78.5                                        # 计算结果
>>> print(eval(input("表达式 = ")))
表达式 = (78 + 82 + 96)//3                  # 输入表达式(78 + 82 + 96)//3,可以带括号
85                                          # 计算结果
```

5.2.7　顺序结构程序设计

1. 赋值语句

赋值语句的语法格式如下：

```
变量名 1,变量名 2,…,变量名 n = 表达式 1,表达式 2,…,表达式 n
```

一条赋值语句可以给一个变量赋值，也可以同时给多个变量赋值，可以赋以常量值，也可以赋以表达式的值。

2. 用 input()函数输入数据

使用 input()函数输入数据的语法格式如下：

```
变量 = input("提示信息")
```

功能：从键盘输入数据并赋给变量，系统把用户的输入看作是字符串。

示例：

```
name = input("请输入姓名：")
age = input("请输入年龄：")
```

系统执行到这样的语句，等待用户输入，用户根据提示信息输入相应的姓名和年龄，如张三和 18，系统都看作是字符串，即 name 的值为"张三"，age 的值为"18"，如果需要，可以用 int()等函数把字符串转换为整数值。

3. 用 print()函数输出数据

使用 print()函数输出数据的语法格式如下：

```
print(表达式 1,表达式 2,…,表达式 n)
```

功能：依次输出 n 个表达式的值，表达式的值可以是整数、实数和字符串，也可以是一个动作控制符，如"\n"表示换行等。其中的表达式可以有一个，可以有多个，多个表达式之间用逗号(,)分开，如果没有任何表达式，print()语句的功能是实现一个换行动作。

4. 顺序结构程序设计

结构化程序设计方法强调程序结构的清晰性，结构化程序由 3 种基本结构组成，分别是顺序结构、分支结构和循环结构。

顺序结构是结构化程序设计中最简单的一种程序结构。在顺序结构程序中，程序的执行是按照语句出现的先后次序顺序执行的，并且每条语句都会被执行到。

【例 5.4】 通过键盘输入圆的半径，计算圆的面积和周长并输出。

问题分析：编写解决该问题的程序要用到 4 个浮点型变量，r 用于存放从键盘输入的圆的半径值，pi 用于存放常量 π 的值，计算出的圆的周长存入变量 peri，圆的面积存入变量 area。

```
# P0504.py
r = float(input("请输入圆的半径值:"))
pi = 3.14
peri = 2 * pi * r
area = pi * r * r
print("周长 = ",peri)
print("面积 = ",area)
```

这是一个典型的顺序结构程序，程序执行时，按书写顺序依次执行程序中的每一条语句。

5.2.8 分支结构程序设计

分支结构又称选择结构。在分支结构中，要根据逻辑条件的成立与否，分别选择执行不同的语句，完成不同的功能。分支结构是通过分支语句来实现的，Python 语言中分支语句包括 if 语句、if-else 语句等。

1. if 语句

if 语句用来实现单分支选择，语法格式为：

```
if 表达式:
  语句块
```

if 语句的执行过程如图 5.8 所示：先计算表达式的值，若值为 True(真)，则执行 if 子句(表达式后面的语句块)，然后执行 if 结构后面的语句；否则跳过 if 子句，直接执行 if 结构后面的语句。

示例：

```
if score<60:
    m=m+1                              # 如果成绩不及格,m 的值加 1
n=n+1                                  # 不管成绩是否及格,n 的值都要加 1
```

如果是对一门课程的考试成绩进行上述操作，其功能是统计参加考试的总人数(n 的值)和不及格人数(m 的值)。

2. if-else 语句

if-else 语句用来实现双分支选择，即 if-else 语句可以根据条件的真(True)或假(False)，执行不同的语句块。

if-else 语句的语法格式为：

```
if 表达式:
  语句块 1
else:
  语句块 2
```

其中，语句块 1 称为 if 子句，语句块 2 称为 else 子句。

if-else 语句的执行过程如图 5.9 所示：先计算表达式的值，若结果为 True，则执行 if 子句(语句块 1)，否则执行 else 子句(语句块 2)。

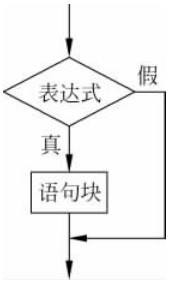

图 5.8　if 分支结构

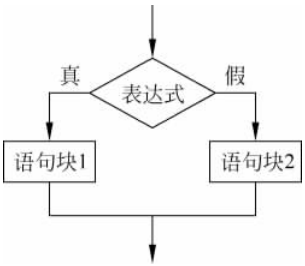

图 5.9　if-else 分支结构

示例：

```
if score<60:
    m=m+1                              # 如果成绩不及格,m 的值加 1
else:
    n=n+1                              # 如果成绩及格,n 的值加 1
```

如果还是对一门课程的考试成绩进行上述操作，其功能是统计不及格人数(m 的值)和及

格人数(n的值)。

5.2.9 循环结构程序设计

完成重复性的工作要用到循环结构程序,循环结构程序通过循环语句来实现,Python语言中有2种循环语句。

1. for循环语句

for循环语句的语法格式为:

```
for 循环变量 in 遍历结构:
    语句块
```

for循环语句的执行过程如图5.10所示:循环变量依次取遍历结构中的值,参与循环体语句块的执行,直至遍历结构中的数据都取完。

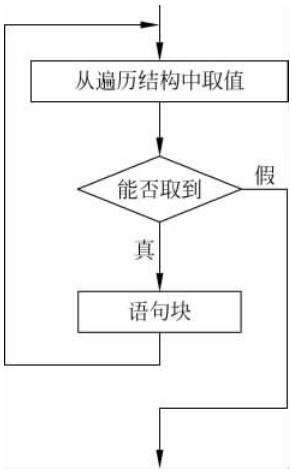

图5.10 for循环结构

【例5.5】 编写程序计算1+2+3+…+100。

问题分析:这是个累加问题,即加法操作要重复多次,可以编写循环程序实现其功能。

```
# P0505.py
sum1 = 0                                  # 累加变量清零
for i in range(1,101):                    # 循环次数为100
    sum1 += i                             # 累加求和
print("sum1 = ",sum1)                     # 输出累加和
```

关于for语句的几点说明如下:

(1) 此处的循环变量i在range(1,101)范围内取值,取值为1～100(不包括101),所以循环体语句执行100次。range()函数的一般格式如下:

```
range(start,end,step)
```

其功能是生成若干个整数值,初始数值为start,结束数值为end－1(注意:不包括end),步长为step。其中start和step都可以省略,省略时默认值分别为0和1。

示例:

```
range(10)                                 # 生成的值为0～9,默认初值为0、步长为1
range(1,10)                               # 生成的值为1～9,默认步长为1
range(1,11,2)                             # 生成的值为1、3、5、7、9,即1～10内的奇数
range(10,0,－1)                           # 生成的值为10～1,步长可以为负数
```

(2) 不同的缩进格式,实现的功能是不一样的。上面的程序段是计算完累加和后输出最后累加和的值,如果改成如下缩进格式,功能变为每累加一次,都会输出中间累加结果:

```
sum1 = 0                                  # 累加变量清零
for i in range(1,101):                    # 循环次数为100
    sum1 += i                             # 累加求和
    print("sum1 = ",sum1)                 # 输出累加和
```

2. while循环语句

while循环语句的语法格式为:

```
while 表达式:
  语句块
```

while 循环语句的执行过程如图 5.11 所示：先计算表达式的值，若表达式的值为真(True)，则执行循环体语句块，然后再次计算表达式的值，若结果仍为真(True)，再次执行循环体语句块，如此继续下去，直至表达式的值变为假(False)，则结束循环体语句块的执行。

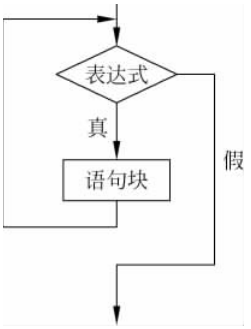

图 5.11　while 循环结构

【例 5.6】 编写程序计算 1＋2＋3＋…，直至累加和大于 5000 为止。

问题分析：这也是个累加问题，但不知道累加次数，只知道累加的结束条件。

```
# P0506.py
sum1 = 0                              # 累加变量清零
i = 1                                 # 设定要累加的初值为 1
while sum1 <= 5000:                   # 循环条件为累加和小于等于 5000
    sum += i                          # 进行累加
    i += 1                            # 要累加的值增 1
print("sum1 = ",sum1)                 # 输出累加和
```

说明：对于已知循环次数的循环结构用 for 语句实现比较简单，对于不知道循环次数但知道结束条件的循环用 while 语句更合适。

3. break 语句

break 语句的语法格式为：

```
break
```

break 语句主要用于循环结构中，其功能是提前结束整个循环，转去执行循环结构后面的语句。

4. continue 语句

continue 语句的语法格式为：

```
continue
```

continue 语句用于循环结构中，其功能是提前结束本次循环，转到循环的开始处判断是否执行下一次循环。

【例 5.7】 从键盘上输入一个正整数，判断其是否为素数。

问题分析：判断一个数 n 是否为素数，就是用 n 逐一除以 2～n/2 的所有整数，如果都不能整除，则确定 n 为素数；如果至少有一个能够整除，则确定 n 不是素数。

```
# P0507_1.py
n = int(input("请输入一个正整数:"))
b = True                              # 设定一个标记值
for i in range(2,n//2 + 1):           # i 的取值范围为 2～n/2
    if (n % i == 0):
        b = False                     # 如果能够整除,则把 b 的值改为 False
if b == True:                         # 如果 b 的值保持为 True,说明都不能整除
    print(n,"是素数")
else:
    print(n,"不是素数")
```

说明：

（1）Python与其他语言最大的区别就是，构成Python程序的代码行必须严格按照缩进的格式规则来书写，Python是通过缩进来识别语句之间的层次关系的。缩进的空格数是可变的，具有相同缩进量的一组语句称为一个语句块。代码的缩进可以通过制表符Tab键或空格键实现，缩进量可多可少，一般设置为4个空格。如果把上述程序改为如下缩进格式，程序功能将有什么变化，自己思考并上机查看输入数值分别为4、17、25时的运行结果：

```
# P0507_2.py
n = int(input("请输入一个正整数:"))
b = True                                    # 设定一个标记值
for i in range(2,n//2 + 1):                 # i的取值范围为2～n/2
    if (n % i == 0):
        b = False
    if b == True:
        print(n,"是素数")
    else:
        print(n,"不是素数")
```

（2）程序P0507_1.py是有改进空间的：只要有一个数能够整除n，就可判断n不是素数，循环就可结束，程序改进如下（增加一个break语句）：

```
# P0506_3.py
n = int(input("请输入一个正整数:"))
b = True                                    # 设定一个标记值
for i in range(2,n//2 + 1):                 # i的取值范围为2～n/2
    if (n % i == 0):
        b = False                           # 如果能够整除，则把b的值改为False
        break                               # 遇到第一个能整除的值，就结束循环
if b == True:
    print(n,"是素数")
else:
    print(n,"不是素数")
```

5.2.10 Python程序实例

【例5.8】 从键盘输入一个字符串，把字符串中的数字字符分离出来并组成一个整数，再乘以数字字符的个数后输出，如果输入“a23TY78hy”，则输出数值9512（2378乘以4）。

问题分析：该程序的关键点是从字符串中截取出各位数字字符并组合成一个整数，需要用到字符串比较、类型转换等操作。

```
# P0508.py
str1 = input("请输入字符串 = ")
cnt = 0
str2 = ""
for ch in str1:
    if ch >= "0" and ch <= "9":
        str2 += ch
        cnt += 1
num = int(str2) * cnt
print(num)
```

说明：Python 中有 3 个逻辑运算符可用，包括逻辑与(and)、逻辑或(or)和逻辑非(not)，逻辑运算的结果是一个逻辑值：True 或 False。

【例 5.9】 判断一个整数是否为回文数。所谓回文数是指一个数的正序和逆序值相等，如 168861、387595783 等。

问题分析：要分析判断的数没有固定的位数，所以只能是逐一分解出个位、十位、百位、……，根据具体数的不同，可能只分解出一位，也可能分解出若干位。

```
# P0509_1.py
n = int(input("n = "))                    # 把键盘输入转化为数值赋给变量 n
m = n                                     # 复制 n 的值给变量 m
s = 0                                     # 赋初值为 0,准备用于存放 n 的逆序值
while m!= 0:                              # 从 m 中逐一分解出个位、十位、百位、……
    k = m % 10                            # 循环处理:第 1 次得到个位值,第 2 次得到十位值,……
    s = s * 10 + k                        # 计算 n 的逆序值
    m = m//10                             # 为分解下一位数做准备
if s == n:                                # 如果 n 的逆序值等于 n 的值
    print(n,"是回文数")
else:
    print(n,"不是回文数")
```

还可以使用对字符串的切片操作完成回文数的判断。

```
# P0509_2.py
num = input("num = ")
if num == num[::-1]:
    print(num,"是回文数")
else:
    print(num,"不是回文数")
```

【例 5.10】 画若干个套在一起的正方形。

问题分析：画正方形图需要用到 Python 的标准库 turtle，turtle 库的主要作用是绘制图形，turtle 库提供了多个用于绘图的画笔控制函数和图形绘制函数。

```
# P0510.py
def draw(x,y,fd):                         # 定义一个绘制正方形的函数
    turtle.goto(x,y)                      # 确定画笔的初始位置
    turtle.pendown()                      # 落下画笔
    for i in range(4):                    # 通过循环画出 4 条边
        turtle.forward(fd)                # 画笔向前移动 fd 个像素位置
        turtle.right(90)                  # 画笔右转 90 度
    turtle.penup()                        # 抬起画笔
import turtle                             # 引入 turtle 库
turtle.setup(500,350,325,175)             # 设置绘图窗口的大小和位置
turtle.pencolor("red")                    # 设置画笔颜色为红色
turtle.pensize(1)                         # 设置画笔尺寸
square_x = -2                             # 第一个正方形开始位置的 x 坐标值
square_y = 2                              # 第一个正方形开始位置的 y 坐标值
length = 5                                # 第一个正方形的边长(像素值)
for k in range(18):                       # 共画 18 个正方形
    draw(square_x,square_y,length)        # 画出一个正方形
    square_x -= 5                         # 画下一个正方形前,x 坐标值减 5
```

```
    square_y += 5                    # y坐标值加5
    length += 10                     # 边长加10
```

程序运行结果如图5.12所示。

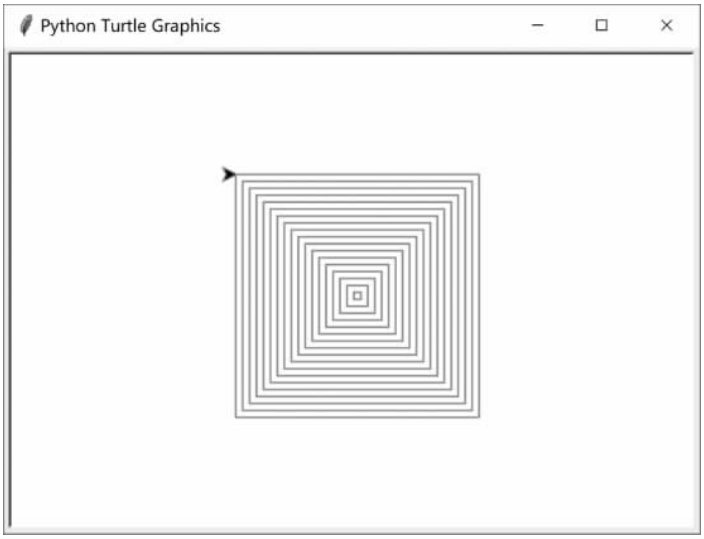

图5.12　正方形图

【例5.11】 根据一名学生每天的时间分配画出饼图。

问题分析：画饼图需要用到第三方库matplotlib。

```
# P0511.py
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams['axes.unicode_minus'] = False
hours = (3,2,8,8,3)
labels = ("吃饭","素质拓展","睡眠","课程学习","娱乐休闲")
colors = ("c","b","m","r","y")
plt.pie(hours,explode = (0,0,0.0,0.06,0),labels = labels,\
    startangle = 90,colors = colors,shadow = True,autopct = '%1.2f%%')
plt.legend()
plt.show()
```

执行该程序画出的饼图如图5.13所示。

【例5.12】 对文本进行分析并生成词云图。

问题分析：对文本进行分析并生成词云图需要用到第三方库jieba、matplotlib和wordloud。

```
# P0512.py
# 引入第三方库 jieba、matplotlib 和 wordloud
import jieba
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
text = ""                                          # 用于存储分词结果
fin = open(r"data.txt", "r")                       # 从文本文件中读取数据
for line in fin.readlines():
```

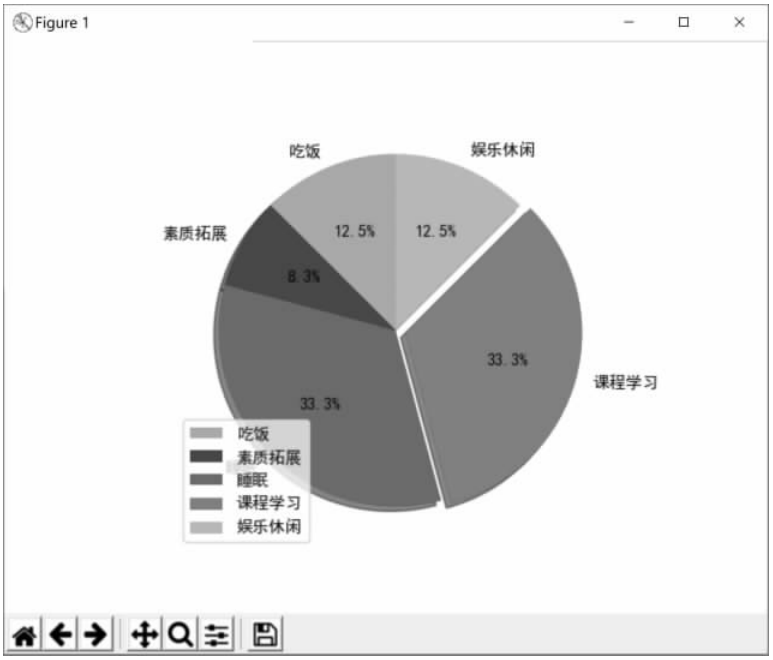

图 5.13　时间分配饼图

```
    line = line.strip("\n")
text += " ".join(jieba.cut(line))                    # 将文本数据分词后存入 text 中
backgroud_Image = plt.imread(r"background.jpg")      # 设置词云背景
# 设置词云样式
wc = WordCloud(
    background_color = "white",                      # 设置背景颜色
    mask = backgroud_Image,                          # 设置背景图片
    font_path = r"C:\Windows\Fonts\SimHei.ttf",      # 设置中文字体
    max_words = 100,                                 # 设置最大现实的字数
    stopwords = STOPWORDS,                           # 设置停用词
    max_font_size = 400,                             # 设置字体最大值
    random_state = 15                                # 设置配色数
)
wc.generate_from_text(text)                          # 生成词云
# 字的颜色来自于背景图片的颜色
wc.recolor(color_func = ImageColorGenerator(backgroud_Image))
plt.imshow(wc)                                       # 绘出词云图
plt.axis("off")                                      # 是否显示 x 轴、y 轴下标
plt.show()                                           # 显示词云图
```

程序运行结果如图 5.14 所示。

说明：Python 提供了大量的内置函数和标准库函数，还有很多的第三方库函数可用，可用于解决各个领域的实际问题。对于标准库函数，需要使用 import 语句引入对应的库后才能使用；对于第三方库函数，需要安装并引入对应的库后才能使用。更多、更深入的了解可查阅专门介绍 Python 程序设计的书籍。

图 5.14 词云图

5.2.11 程序设计风格

早期的程序设计，由于计算机速度比较慢和存储容量比较小，在实现功能的基础上，强调的是效率第一，比较注重编程技巧。但随着计算机性能的提高及程序规模的逐渐变大，这种模式的缺点日渐明显，最主要的是程序难以阅读和理解、难以找到程序中存在的错误、难以改正程序中的错误，这种缺点有时是致命的，耗时耗力编写出来的程序无法投入使用，只能重新编写。就如同没有资质的包工队用盖平房的模式去建几十层的高楼大厦，即使勉强盖起来了，也由于各种质量问题而无法投入使用，只能炸掉重建。

人们在总结较大规模程序设计经验教训的基础上，提出了保证程序质量的程序设计规范。在编程实践中逐步形成良好的程序设计风格，这对有志于从事程序设计和软件开发的计算专业人员来说是至关重要的。

对于编写较大规模的程序，在实现功能的基础上，强调的是清晰第一，即编写的程序要易于理解、易于找到程序中存在的错误、易于改正错误。基于此，良好的程序风格主要包括如下几个方面。

(1) 标识符的命名要风格统一、见名知义。

(2) 一般一行写一条语句，一条长语句可以写在多行上，但尽量不要把多条语句写在一行上。

(3) 采用缩进格式，即同一层次的语句要对齐，内层语句要缩进若干个字符，这样能够比较清楚地表达出程序的结构，增加程序的可读性。

(4) 适当书写注释信息，注释是对语句所做的说明，有助于阅读者对程序的理解。

(5) 尽量少用 goto 语句，使用 goto 语句虽能增加程序的灵活性，但使用多了容易导致程序结构混乱。

本章给出的程序示例尽量符合上述要求，阅读程序时可以体会良好的程序设计风格的作用。

5.2.12　算法设计与分析

用计算机解决问题的过程，可以分成如下几个阶段。

（1）分析问题、设计算法：认真分析要解决的问题及要实现的功能，给出解决问题的明确步骤，即设计出针对要解决问题的算法。

（2）选定语言、编写程序：根据问题的性质，选定一种合适的程序设计语言（及相应的开发环境），依据设计出的算法编写源程序。

（3）编译：对编写出的源程序进行编译，程序中如果没有错误的话，则编译生成目标文件（与源程序对应的机器语言文件）。如果发现程序中有错误，设法找到错误并改正。

（4）连接：对生成的目标文件进行连接操作，把目标文件与相应的环境（运行系统、函数库）连接成可执行的程序。

（5）调试执行：执行可执行程序，选用一些有代表性的数据对程序进行测试，经过一定的测试，如果没有发现错误，程序就可以交付使用了。如果在测试中发现错误，就要分析错误的性质。如果是算法设计有问题，就应重新分析问题、修改算法或重新设计算法；如果是程序编写有问题，就设法在程序中找到错误所在并改正（程序查错排错）。对于较大规模的程序，程序查错排错是一项困难的工作，既需要经验，也需要一定的方法和工具支持。

对于写文章来说，初学者感觉遣词造句是困难的，但写出好文章真正的难点在于文章的总体构思和创意。编写程序也一样，初学者的难点在于语言基本要素和语法规则的掌握，而真正设计出高水平程序的基础是良好的算法设计，相对来说，有了好的算法，再编写程序就简单了。

我国针对计算机专业人员的水平考试，主要有三个级别：程序员、高级程序员和系统分析师。

程序员（programmer）相当于助理工程师，主要工作是编写功能比较单一的程序，并完成模块程序的调试、测试工作。要具备计算机软硬件的基础知识，熟练掌握某种程序设计语言及相应的开发环境，只要给出了明确的算法，就能编写程序实现。

系统分析师（systems analyst）也称为系统分析员，相当于高级工程师，可以担任项目组长或项目经理，主要工作是进行软件开发项目的总体分析和设计工作，了解计算机软硬件技术的最新发展，理解客户需求和项目的业务流程，具备比较强的需求分析能力、整体框架构建能力、流程处理能力、模块分解能力、整体项目评估能力和团队组织管理能力。

高级程序员（senior programmer）相当于工程师，其承担的工作介于程序员和系统分析师之间，一般是协助系统分析师完成系统分析和算法设计工作，或组织程序员完成大型模块的开发工作。

程序员的主要工作是编写程序，高级程序员、系统分析师的主要工作是设计算法和软件系统的总体设计，从中也可以看出算法设计的重要性。计算机专业的大学毕业生，可以沿着程序员、高级程序员、系统分析师的层次发展。

1. 程序与算法

现实生活中，做任何事情都需要经过一定的步骤才能完成。例如，教师按教学计划授课、工程师设计施工方案等都必须按照一定的步骤进行。

为解决一个问题而采取的方法和步骤，称为算法。算法（algorithm）是被精确定义的一组规则，规定先做什么，再做什么，以及判断某种情况下做哪种操作；或说算法是步进式的完成任务的过程。

程序(program)指为让计算机完成特定的任务而设计的指令序列或语句序列，一般认为机器语言程序或汇编语言源程序由指令序列构成，高级语言源程序由语句序列构成。程序设计(programming)是沟通算法与计算机的桥梁；程序是程序设计人员编写的、计算机能够理解并执行的一些命令的集合，是解决问题的具体算法在计算机中的实现。

2. 算法的特点

算法反映解决问题的步骤，不同的问题需要用不同的算法来解决，同一问题也可能有不同的解决方法，但是一个算法必须具有以下特性。

(1) 有穷性。一个算法必须总是在执行有限个操作步骤和可以接受的时间内完成其执行过程。即对于一个算法，要求其在时间和空间上均是有穷的。

(2) 确定性。算法中的每一步都必须有明确的含义，不允许存在二义性。

(3) 有效性。算法中描述的每一步操作都应该能有效地执行，并最终得到确定的结果。

(4) 输入及输出。一个算法应该有零个或多个输入数据，有一个或多个输出数据。执行算法的目的是为了求解，而“解”就是输出，因此没有输出的算法是没有意义的。

3. 算法的表示

1) 用自然语言表示

自然语言就是人们日常使用的语言，可以是中文、英文等。

例如，求三个数的最大值的问题，可以用中文描述为先比较前两个数，找到大的那个数，再让其与第三个数进行比较，找到两者中大的数即为所求。

2) 用传统流程图表示

传统流程图是用规定的一组图形符号、流程线和文字说明来表示各种操作的算法表示方法。图5.11就是一个简单的流程图。

3) 用伪码表示

伪码是用一种介于自然语言和计算机语言之间的文字和符号来描述算法。接近计算机语言，便于向计算机程序过渡。比计算机语言形式灵活、格式紧凑，没有严格的语法格式。是目前用得比较多的一种算法表示形式。

4. 算法的评价标准

用计算机解决问题的关键是算法的设计，对于同一个问题，可以设计出不同的算法，如何评价算法的优劣是算法分析、比较、选择的基础。目前，可以从正确性、时间复杂性、空间复杂性和可理解性4个方面对算法进行评价。

1) 算法的正确性

算法的正确性指算法能够正确地完成所要解决的问题，就目前的研究来看，要想通过理论方式证明一个算法的正确性是非常复杂和困难的，一般采用测试的方法，基于算法编写程序，然后对程序进行测试。针对所要解决的问题，选定一些有代表性的输入数据，经程序执行后，查看输出结果是否和预期结果一致，如果不一致，则说明程序中存在错误，应予以查找并改正。经过一定范围的测试和程序改正，不再发现新的错误，程序可以交付使用，在使用过程中仍有可能发现错误，再继续改正，这时的改正称为程序维护。

例如，一个对考试成绩进行管理的程序，主要功能是按学生或按课程查询成绩，对学生按考试成绩排名等。如果有100个学生，你可以选择第1名、第10名、第50名、第90名、最后一名学生的成绩进行计算，看计算结果和手工计算结果是否一致。这比简单地选择前5个学生的成绩进行测试更有代表性，更有可能发现程序中的错误。

一些大的软件开发公司开发的软件，先是在开发人员内部进行测试，然后在公司内部（与开发人员不同的一些人）进行测试，最后再请一些公司外的用户进行测试。

对于小程序来说，测试工作是比较简单的，如考试成绩管理程序，可能有半天的时间就能完成测试工作。对于大型的程序，可能需要数月的测试时间，即使投入实际应用后，也还会在使用中发现错误、改正错误。

2）算法的时间复杂度

时间复杂度指依据算法编写出程序后在计算机上运行时所耗费的时间度量。一个程序在计算机上运行的时间取决于程序运行时输入的数据量、对源程序编译所需要的时间、执行每条语句所需要的时间及语句重复执行的次数等。其中，最重要的是语句重复执行的次数。通常，把整个程序中语句的重复执行次数之和作为该程序的时间复杂度，记为 $T(n)$，其中的 n 为问题的规模。对于一个从线性表中查找某个数据的算法，n 为线性表的长度，即线性表中数据的个数。

算法的时间复杂度 $T(n)$实际上是表示当问题的规模 n 充分大时，该程序运行时间的一个数量级，用 O 表示。比较两个算法的时间复杂度时，不是比较两个算法对应程序的具体执行时间，这涉及编程语言、编程水平和计算机速度等多种因素，而是比较两个算法相对于问题规模 n 所耗费时间的数量级。

例如，比较线性表的顺序查找和折半查找算法。对于顺序查找算法，由于其平均查找次数为 $n/2$（查找语句重复执行 $n/2$ 次），所以其时间复杂度为 $O(n)$，$n/2$ 和 n 是一个数量级；而折半查找的查找次数为 $\mathrm{lb}n$（查找语句重复执行 $\mathrm{lb}n$ 次），所以折半查找算法的时间复杂度为 $O(\mathrm{lb}n)$。相对于顺序查找，n 越大，折半查找的速度优势越明显，但折半查找的基础是线性表中的数据要有序。

3）算法的空间复杂度

空间复杂度指，依据算法编写出程序后在计算机上运行时所需内存空间大小的度量，也是和问题规模 n 有关的度量。

4）算法的可理解性

算法是为了人们的阅读与交流，可理解性好的算法有利于人们的正确理解，有利于程序员据此编写出正确的程序。

5.3　数 据 结 构

在计算机发展的早期，编写程序的目的是为了完成对数值数据的计算。随着计算机技术的不断发展和应用范围的拓展，数据处理成为计算机的一个重要应用领域，此时的数据包括数值数据，也包括非数值数据（如字符串、图像等），处理既可以是算术运算，也可以是插入、删除、查找和排序等操作。

程序要处理的数据需要存储在内存单元中，已知，整个内存空间是由连续编址的一个个内存单元组成的，就是说，多么复杂的数据也只能存放在这样的一个空间内，这是数据存储的物理结构。编程人员直接面对这种存储方式，对于一些简单的运算是可以的，实际上在机器语言和汇编语言编程阶段，程序员就是直接使用这种物理存储结构。这种方式数据处理能力有限、编程复杂，对于矩阵、家族关系表这样的数据就会增加编程人员的编程难度和工作量。能否找到一种机制，使得数据的内部存储结构（物理结构）是线性连续的，而其逻辑结构更符合人们习

惯的方式，编写程序时面对的是逻辑结构（数据的一种抽象表示），程序执行时，由支持相应结构的编译程序自动把逻辑结构映射成物理结构，从而简化程序的编写，减轻编程人员的工作量，这就是数据结构要解决的问题之一。

5.3.1 概念和术语

数据（data）是信息的载体，它能够被计算机识别、存储和加工处理。计算机科学中，所谓数据就是计算机加工处理的对象，它可以是数值数据，也可以是非数值数据，如图像、声音、文本等。

数据项（data item）是数据不可分割的最小单位。数据项有名和值之分，数据项名是数据项的标识，用变量定义，而数据项值是它的一个可能取值。年龄就是一个数据项，在 Python 语言中可以定义为变量 age，其取值可以是 19、20、21 等。

数据元素（data element）是数据的基本单位，具有完整、确定的实际意义。在不同的条件下，数据元素又可称为元素、站点、顶点、记录等。数据元素一般由若干数据项组成。学生的基本情况就是一个数据元素，由学号、姓名、性别、年龄等数据项组成。

数据对象或称数据元素类（data object），是具有相同性质的数据元素的集合，是数据的一个子集。在某个具体问题中，数据元素都具有相同的性质，属于同一数据对象，数据元素是数据元素类的一个实例。对于一个学生管理系统来说，某大学所有学生的基本情况就是数据，所有本科生的基本情况、所有硕士生的基本情况可以看作是不同的数据对象。

数据结构（data structure）指互相之间存在着一种或多种关系的数据元素的集合。在任何问题中，数据元素都不是孤立的，它们之间存在着这样或那样的关系（联系），这种数据元素之间的关系称为结构。

根据数据元素间关系的不同特性，通常分成如下三类基本结构。

（1）线性结构（linear structure）。数据元素之间存在着一对一的关系，如线性表、栈、队列和数组等。按学号排列的学生数据可以看作是一个线性表，每个学生有一个且只有一个前驱（第一个学生除外），每个学生有一个且只有一个后继（最后一个学生除外）。

（2）树形结构（tree structure）。数据元素之间存在着一对多的关系，如树、二叉树和森林等。一个单位的工作人员之间的关系就可以表示成一棵树，每个人只有一个直接领导（单位的最高领导除外），有多个直接下属（最基层的工作人员除外）。

（3）图状结构（graph structure）。数据元素之间存在着多对多的关系，图状结构也称网状结构，如无向图和有向图等。铁路交通图是一种典型的图状结构，任意两个城市之间可能存在多条路径连通。

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构可以看作是从具体问题抽象出来的数学模型，描述的是数据元素之间的逻辑关系。数据在计算机中的表示称为数据的物理结构或存储结构，它研究数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。

数据的存储结构可采用顺序存储或链式存储的方法。

顺序存储方法是把逻辑上相邻的元素存储在物理位置也相邻的存储单元中，由此得到的存储表示称为顺序存储结构。顺序存储结构常借助于程序设计语言中的数组来实现。

链式存储方法对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示，由此得到的存储表示称为链式存储结构。链式存储结构通常借助于程序

设计语言中的指针来实现。

借助于列表介绍线性结构的存储及应用，对于树形结构和网状结构只作简要说明，体会一下数据结构对程序设计的作用，详细内容可以在数据结构课程中学习。

5.3.2　线性结构

线性结构是最常用、最简单的数据结构，包括线性表、队列、栈等，C 语言中的数组、Python 语言中的列表和元组都是线性结构。线性表、队列、栈可以用数组或列表来实现，本节以 Python 语言中的列表为例介绍线性结构的使用。

列表(list)是包含 0 个或多个数据的有序序列，其中的每个数据称为元素，列表的元素个数(列表长度)和元素内容都是可以改变的。使用列表，能够灵活方便地对批量数据进行组织和处理。

1. 创建列表

创建列表的语法格式如下：

```
列表名 = [值 1,值 2,值 3,…,值 n]
```

功能：把一组值放在一对方括号内组织成列表值并赋值给一个列表变量。列表值可以有 0 个、1 个或多个，如果有多个列表值，值与值之间用逗号分隔。

示例：

```
>>> list1 = [78,62,93,85,68]
>>> list2 = ["2018001","张三","男",19,"金融学"]
>>> list3 = [36]
>>> list4 = []
```

也可以通过 list()函数创建列表，例如：

```
>>> list6 = list(range(1,6))          # 等价于 list6 = [1,2,3,4,5]
>>> list7 = list("Python 程序")       # 等价于 list7 = [ 'P','y','t','h','o','n','程','序']
```

2. 访问列表

列表创建后，就可以访问使用。对列表的访问，除了可以整体赋值外，常用的方式是访问其中的元素，语法格式如下：

```
列表名[索引值]
```

列表是序列类型，其中的元素按所在的位置顺序都有一个唯一的索引值(序号)，通过这个索引值可以访问到指定的元素。系统为列表元素设置了两套索引：正向索引和逆向索引，正向索引取值为 0、1、2、……，分别对应第 1、第 2、第 3 个元素等，逆向索引取值为－1、－2、－3、……，分别对应倒数第 1、倒数第 2、倒数第 3 个元素等。需要注意的是正向索引值从 0 开始递增，逆向索引值从－1 开始递减。

对于 list2＝["2018001","张三","男",19,"金融学"]，系统为其设定的索引值如图 5.15 所示。

```
list2[0]与 list2[－5]的值相同，为字符串"2018001"
list2[3]与 list2[－2]的值相同，为整数 19
list2[－1]与 list2[4]的值相同，为字符串"金融学"
```

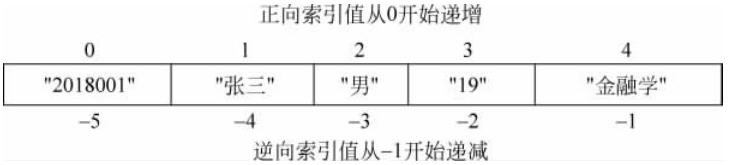

图 5.15　列表的索引值

如果列表中不存在与给定索引值对应的元素，系统给出相应的错误提示信息。

3. 更新列表

列表创建后，其中的元素值是可以修改的，除此之外，还可以向列表中增加元素和删除列表中的已有元素，即列表的长度也是可以改变的。

增加列表元素的语法格式如下：

```
列表名.append(新增元素值)
```

或

```
列表名.insert(索引值,新增元素值)
```

其中，append()函数用于在列表的末尾追加元素，insert()函数用于在列表的指定位置插入元素。如果有多个元素需要插入列表，可以先用 append()函数追加到列表的末尾，再进行排序，这样能节省时间。

示例：

```
>>> list1 = [2,3,5,9,11]                # 创建列表,初值为[2,3,5,9,11]
>>> list1.append(13)                    # 在列表的尾部追加数值 13,列表值变为[2,3,5,9,11,13]
>>> list1.insert(3,7)                   # 在指定位置插入数值 7,列表值变为[2,3,5,7,9,11,13]
```

删除列表元素的语法格式如下：

```
列表名.remove(元素值)
```

或

```
del 列表名[索引值]
```

或

```
del 列表名
```

其中，remove()函数用于从列表中删除指定的值，若有多个值和指定值相同，只删除第一个。del 格式用于删除和指定索引值对应的列表元素值或删除整个列表。

示例：

```
>>> list1.remove(11)                    # 删除列表中的值 11
>>> del list1[2]                        # 删除列表中索引值为 2 的元素(索引值从 0 开始)
```

修改列表元素值的语法格式如下：

```
列表名[索引值] = 新元素值
```

通过赋值语句的形式，用新元素值替换指定位置的现有元素值。

示例：

```
>>> list2 = ["2018001","小明","男",19,"数学"]          # 创建列表
>>> list2[1] = "张小明"                                # 修改索引值为 1 的元素值
>>> list2                                              # 显示修改后的列表值
['2018001', '张小明', '男', 19, '数学']
>>> list2[-1:] = ["金融学","二班"]                     # 修改最后一个元素值
>>> list2                                              # 显示修改后的列表值
['2018001', '张小明', '男', 19, '金融学', '二班']
>>> list2[1:3] = ["张明"]                              # 修改索引值为 1 和 2 的元素值
>>> list2                                              # 显示修改后的列表值
['2018001', '张明', 19, '金融学', '二班']
```

说明：

(1) 对于已创建列表，既可以修改某个指定索引值对应的元素值，也可以修改指定索引值范围内的若干元素值，如果给定的新值个数少于指定范围内的元素值个数(如 list2[1:3]=["张明"])，则相当于删除元素值，如果给定的新值个数多于指定范围内的元素值个数(如 list2[-1:]=["金融学","二班"])，则相当于增加元素值。

(2) list2[i:j]指列表中的第 i 个至第 j-1 个元素，不包括第 j 个元素，正向索引值从 0 开始递增，逆向索引值从-1 开始递减；如果省略 i，默认从 0 开始，如果省略 j，默认到最后一个元素结束，包括最后一个元素。

对于批量数据处理，列表有着强大灵活的功能，列表的常用操作见表 5.3，有些操作通过运算符实现，有些操作通过 Python 内置函数实现。

表 5.3　常用列表操作

操作符(运算符或函数名)	示例与功能描述
+	li1+li2：连接两个列表
*	li * n 或 n * li：将列表自身连接 n 次
in	x in li：如果 x 是 li 的元素，返回 True，否则返回 False
not in	x not in li：如果 x 不是 li 的元素，返回 True，否则返回 False
[]	li[i]：定位列表中索引值为 i 的元素
[::]	li[i:j:k]：切片操作，返回列表中索引值从 i 开始，到 j-1 结束(不包括 j)，步长为 k 的若干个元素组成的列表。省略 i 时默认从 0 开始；省略 j 时默认到最后一个元素结束，包括最后一个元素；省略 k 时默认步长值为 1，此时可同时省略 k 前面的冒号
len(列表名)	len(li)：返回列表 li 的长度，即列表 li 中的元素个数
max(列表名)	max(li)：返回列表 li 中的最大元素
min(列表名)	min(li)：返回列表 li 中的最小元素
sorted(列表名，reverse=False/True)	sorted(li)：返回一个对 li 列表排好序的新列表，列表 li 中元素的顺序不变。当第 2 个参数取值 False 时可省略，此时按升序排列；取值为 True 时按降序排列，不可省略
reversed(列表名)	reversed(li)：返回一个对列表 li 进行逆序操作后的迭代器，需要用 list(reversed(li))形式转换为列表
sum(列表名)	sum(li)：如果列表中都是数值型元素，返回累加和

注：表中 li、li1、li2 都是已存在的列表名。

【例 5.13】 统计一批数据中奇数的个数。

问题分析：把一批数据存入一个列表变量，通过逐一访问列表中的值，统计出奇数的个数。

```
# P0513.py
data_list = [78,81,92,67,93]                    # 创建数据列表
num = 0                                         # 计数器清 0
for i in range(0,5):                            # 遍历列表中每一个数据
    if data_list[i] % 2 == 1:                   # 统计奇数的个数
        num += 1
print("num = ",num)                             # 输出结果
```

【例 5.14】 输出杨辉三角形前 n 行的数据，n 的值由用户输入。

问题分析：6 行的杨辉三角形如图 5.16 所示，从图中可以看出，第 1 列的值全为 1，行、列号相同的位置（如第 2 行的第 2 列、第 3 行的第 3 列等），其值也全为 1，其他位置的值等于正上方位置值和左上方位置值之和（如第 6 行第 3 列的值等于第 5 行第 3 列的值加上第 5 行第 2 列的值）。可以把整个杨辉三角形的数据存入一个列表（yanghui_list），其中的每个元素（代表三角形中每行的数据）也是一个列表（row_list）。在程序中，逐行生成数据，并存入行列表 row_list 和三角形列表 yanghui_list。

```
# P0514.py
yanghui_list = [[1],[1,1]]                      # 创建三角形列表,并设定前两行的值
n = int(input("n = "))                          # 输入要输出的杨辉三角形的行数
for i in range(2,n):                            # 生成第 3 行至第 n 行的值
    row_list = list()                           # 创建初值为空的行列表
    for j in range(i + 1):                      # 设定行列表中各元素的值为 0
        row_list.append(0)
    row_list[0] = 1                             # 改第 1 列的值为 1
    row_list[i] = 1                             # 改第 i + 1 列的值为 1,该行为第 i + 1 行
    for k in range(1,i):                        # 改本行中第 2 至第 i 列的值
        row_list[k] = yanghui_list[i - 1][k - 1] + yanghui_list[i - 1][k]
    yanghui_list.append(row_list)               # 把一行数据作为元素追加到三角形列表
for i in range(n):                              # 用二重循环输出三角形列表的值
    for k in range(i + 1):
        print(yanghui_list[i][k],end = "\t")
    print("\n")
```

说明：如果一个列表中的元素也是列表，其实表示的是二维表或二维数组，即对于其他高级语言中的二维数组，在 Python 中可以用元素为列表的列表表示。对于二维表数据，本书中用第 1 行、第 2 行、第 3 行、……，第 1 列、第 2 列、第 3 列、……，来表示，但要注意正向索引值是从 0 开始。对于如图 5.17 所示的二维表数据，可以定义为如下列表：

```
data_list = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
```

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

图 5.16 杨辉三角形数据（6 行）

```
1  2  3  4
5  6  7  8
9 10 11 12
```

图 5.17 二维表数据

data_list[0][0]表示第 1 行第 1 列数据(其值为 1)、data_list[2][3]表示第 3 行第 4 列数据(其值为 12)。

5.3.3　树形结构

线性结构的特点是逻辑结构简单,易于进行查找、插入和删除等操作,主要用于对具有单一的前驱和后继的数据关系进行描述,而现实环境中数据元素之间的关系也有很多是非线性的,如人类社会的族谱、各种社会机构的组织形式等具有明显的层次关系,用非线性的树形结构描述更为合适。

1. 树

树(tree)是 $n(n\geqslant 0)$个结点的有限集合。当 $n=0$ 时,称为空树。在一棵非空树 T 中:

(1) 有且仅有一个特定的结点称为树的根结点;

(2) 当 $n>1$ 时,除根结点之外的其余结点被分成 $m(m\geqslant 1)$个互不相交的集合 $T_1,T_2,\cdots,T_m$,其中每一个集合 $T_i(1\leqslant i\leqslant m)$本身又是一棵树,并且称为根结点的子树。

在树中,一个结点可以看作是一个数据元素。图 5.18 是一棵具有 11 个结点的树,根结点为 A。

如果一棵树中结点的各子树从左到右是有次序的,即若交换了某结点各子树的相对位置,则构成不同的树,称这棵树为有序树;反之,则称为无序树。零棵或有限棵不相交的树的集合称为森林(forest)。任何一棵树,删去根结点就变成了森林。如图 5.19 所示是由图 5.18 的树去掉根结点后形成的森林。

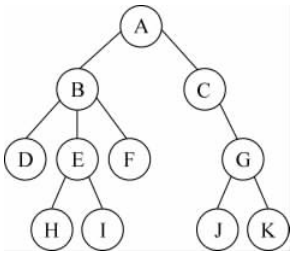

图 5.18　树

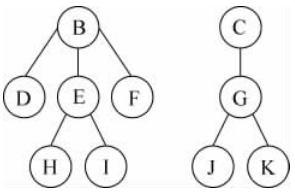

图 5.19　森林

结点所拥有的子树的个数称为该结点的度,度为 0 的结点称为叶结点,度不为 0 的结点称为分支结点。一棵树的结点除叶结点外,其余的都是分支结点。树中一个结点的子树的根结点称为这个结点的孩子(子结点),这个结点称为它的子结点的父结点,具有同一个父结点的子结点互称为兄弟结点。

如果一棵树的一串结点 $n_1,n_2,\cdots,n_k$ 有关系:结点 n_i 是 n_{i+1}的父结点$(1\leqslant i<k)$,就把 $n_1,n_2,\cdots,n_k$ 称为一条由 $n_1\sim n_k$ 的路径。这条路径的长度是 $k-1$。在树中,如果有一条路径从结点 M 到结点 N,那么 M 就称为 N 的祖先,而 N 称为 M 的子孙。

规定树的根结点的层数为 1,其余结点的层数等于它的父结点的层数加 1。树中所有结点的最大层数称为树的深度,树中各结点度的最大值称为该树的度。

2. 二叉树

1) 二叉树的概念

二叉树(binary tree)是有限个结点的集合,该集合或为空或由一个称为根的结点及两个不相

交的、被分别称为左子树和右子树的二叉树组成。当集合为空时，称该二叉树为空二叉树。

二叉树是有序的，即使树中结点只有一棵子树，也要区分它是左子树还是右子树。因此二叉树具有5种基本形态：空二叉树、只有根结点的二叉树、只有根结点及其左子树的二叉树、只有根结点及其右子树的二叉树、左右子树都有的二叉树，如图5.20所示。

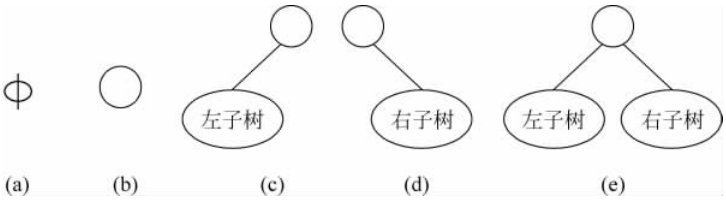

图5.20 二叉树的5种基本形态

在二叉树中，左子树的根称为左孩子，右子树的根称为右孩子。如果所有分支结点都存在左子树和右子树，并且所有叶子结点都在同一层上，这样的一棵二叉树称作满二叉树。

一棵深度为 k 的有 n 个结点的二叉树，对树中的结点按从上至下、从左到右的顺序进行编号，如果编号为 $i(1\leqslant i\leqslant n)$ 的结点与满二叉树中编号为 i 的结点在二叉树中的位置相同，则称这棵二叉树为完全二叉树。完全二叉树的特点：叶子结点只能出现在最下层和次最下层，且最下层的叶子结点集中在树的左部。显然，一棵满二叉树必定是一棵完全二叉树，而完全二叉树未必是满二叉树。

图5.21是一棵满二叉树，当然也是完全二叉树；图5.22是完全二叉树，但不是满二叉树；图5.23不是完全二叉树，当然也就不是满二叉树。

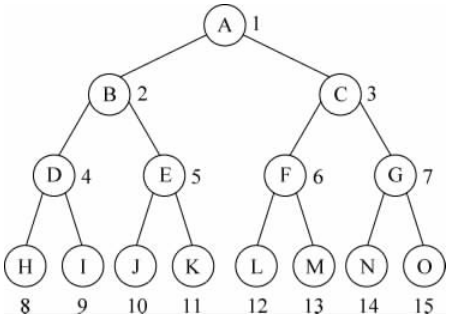

图5.21 满二叉树

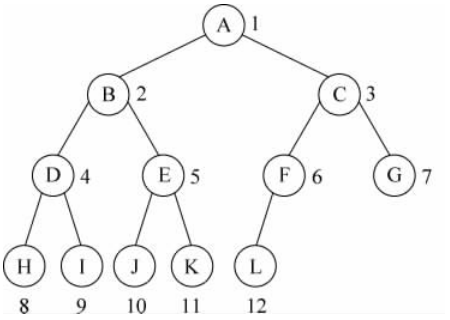

图5.22 完全二叉树

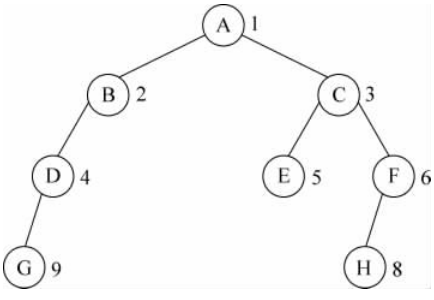

图5.23 非完全二叉树

2）二叉树的主要性质

性质1：一棵非空二叉树的第 i 层上最多有 2^{i-1} 个结点（$i\geqslant 1$）。

性质2：一棵深度为 k 的二叉树中，最多具有 2^k-1 个结点。

性质3：具有 n 个结点的完全二叉树的深度 k 为 $[\mathrm{lb}n]+1$。

Python语言中可以用列表或字典来表示二叉树。

3. 字典

一批数据存入列表，查找或读取、修改某个数据元素时，需要给出该数据元素的索引值，数据量比较大时，记住数据元素的索引值不是一件容易的事情。如果能够按照某个关键字的值（学号、身份证号等）查找或读取批量数据中的信息，则对数据的操作更为简单方便。以字典方式组织数据就可以实现按关键字查找和读取、修改信息。

1）创建字典

创建字典的语法格式如下：

```
字典名 = {键 1:值 1,键 2:值 2,键 3:值 3,…,键 n:值 n}
```

功能：字典也是由若干个元素组成，由一对大括号括起来，每个元素是一个“键-值”对的形式，“键-值”对之间用逗号分开。如果有多个“键”相同的“键-值”对，只保留最后一个。

示例：

```
>>> dic1 = {"数学":78,"语文":82,"英语":67,"计算机":91}      # 课程名与成绩
>>> dic2 = {"小明":"数学","小花":"英语","小莲":"金融"}      # 学生姓名与专业
>>> dic3 = {}                                               # 创建一个空字典
```

2）访问字典

和列表不同，字典是一个无序序列，其中的元素没有对应的索引值，元素的存储顺序（以及对应的显示顺序）可能与创建字典时的书写顺序不一致。对字典的访问是根据“键”来找对应的“值”，语法格式如下：

```
字典名[键]
```

示例：

```
>>> score = dic1["计算机"]                                  # 获取"计算机"课程的考试成绩
>>> specialty = dic2["小莲"]                                # 获取"小莲"的所学专业
```

可对字典进行操作的函数和方法如表 5.4 所示。

表 5.4　字典操作方法和函数

函数/方法	示例与功能描述
字典名.keys()	dic.key()：返回指定字典的所有“键”
字典名.values()	dic.values()：返回指定字典的所有“值”
字典名.items()	dic.items()：返回指定字典的所有“键-值”对
字典名.get(键,默认值)	dic.get(key,default)：存在与 key 相同的“键”，则返回相应的“值”，否则返回 default
字典名.pop(键,默认值)	dic.pop(key,default)：存在与 key 相同的“键”，则返回相应的“值”，同时删除“键-值”对，否则返回 default
字典名.popitem()	dic.popitem()：随机从字典中取出一个“键-值”对，以元组（键,值）形式返回，该“键-值”对从字典中删除
字典名.clear()	dic.clear()：删除指定字典的所有“键-值”对，变为空字典
del 字典名[键]	del dic[key]：删除字典中“键”为 key 的“键-值”对
键 in 字典名	key in dic：存在与 key 相同的“键”，则返回 True，否则返回 False

注：dic 是已存在的字典名。

3）更新字典

使用赋值语句可以增加元素或修改现有元素的值，语法格式如下：

字典名[键] = 值

如果在字典中没有找到指定的“键”，则在字典中增加一个“键-值”对，如果找到，则用指定的“值”替换现有值。该语句既能增加元素，又能修改元素值。使用此功能要仔细，否则，本来要进行修改操作，由于“键”没写对，实际是完成了增加元素的功能。

示例：

```
>>> dic1 = {"数学":78,"语文":82,"英语":67,"计算机":91}
>>> dic1["英语"] = 76                # 修改"英语"成绩为 76
>>> dic1
{'语文': 82, '英语': 76, '计算机': 91, '数学': 78}
>>> dic1["法语"] = 76                # 如果修改时把"英语"误写为"法语"
>>> dic1                             # 等同于增加了一个元素("法语":76)
{'法语': 76, '语文': 82, '英语': 67, '计算机': 91, '数学': 78}
```

还可以使用 update()函数进行字典的合并，语法格式如下：

字典名 1.update(字典名 2)

如果两个字典的“键”没有相同的，则把字典 2 的“键-值”对添加到字典 1 中（实现两个字典的合并），如果有相同的，用字典 2 中的值修改字典 1 中相同“键”的对应“值”。

示例：

```
>>> dic1 = {"数学":78,"语文":82,"英语":67,"计算机":91}
>>> dic2 = {"英语":76,"物理":86}
>>> dic1.update(dict2)
>>> dic1
{'语文': 82, '英语': 76, '数学': 78, '物理': 86, '计算机': 91}
```

删除元素与删除字典的语法格式如下：

del 字典名[键]

如果在字典中找到指定的“键”，则删除“键”和对应的“值”，如果没有找到指定的“键”，则会报错。如果只有字典名，则删除整个字典。

还可以使用 pop()函数删除字典元素，语法格式如下：

字典名.pop(键,值)

如果字典中存在指定的“键”，则返回对应的“值”，同时删除该“键-值”对，如果指定的“键”不存在，返回函数中给出的“值”。例如：

```
>>> dic1 = {"数学":78,"语文":82,"英语":67,"计算机":91}
>>> dic1.pop("数学":60)              # 由于存在"数学",函数返回值为 78,删除键-值对"数学":78
>>> dic1.pop("化学":60)              # 由于不存在"化学",函数返回值为 60
```

【例 5.15】 基于字典实现学生信息管理。

问题分析：可以把学生信息组织成一个字典，以学号为“键”，其他信息为“值”，但这个“值”又是一个字典，在这种数据组织方式下，可以方便地实现查找、统计、修改等功能，如下程

序实现了按输入查找某个专业的学生信息的功能。

```
# P0515.py
students = {
    "2018001":{ "姓名":"小明","性别":"男","年龄":18,"专业":"数学" },
    "2018002":{ "姓名":"小花","性别":"女","年龄":19,"专业":"英语" },
    "2018006":{ "姓名":"小莲","性别":"女","年龄":18,"专业":"数学" },
    "2018009":{ "姓名":"小亮","性别":"男","年龄":20,"专业":"化学" }
}
specialty = input("请输入要查找的专业:")
for stu_number,stu_info in students.items():
    if stu_info["专业"] == specialty:                    #查找所有指定专业的学生
        print(stu_number,end = " ")
        print(stu_info["姓名"],stu_info["性别"],stu_info["年龄"],stu_info["专业"])
```

采用字典方式存储数据的优点是：如果字典的结构有所变化，增加或减少了“键-值”对，程序仍能执行并得到正确结果。

4. 二叉树的存储

1）顺序存储结构

二叉树的顺序存储，是用一组连续的存储单元（数组）存放二叉树中的结点。一般是按照二叉树结点从上至下、从左到右的顺序存储。这样结点在存储位置上的前驱后继关系并不一定是它们在逻辑上的邻接关系（如父子结点关系），只有通过一些方法能够确定某结点在逻辑上的前驱结点（如父结点）和后继结点（如子结点），这种存储才有意义。因此，依据二叉树的性质，完全二叉树和满二叉树采用顺序存储比较合适，树中结点的序号可以唯一地反映出结点之间的逻辑关系，这样既能够最大可能地节省存储空间，又可以利用列表元素的索引值确定结点在二叉树中的位置以及结点之间的关系。图 5.22 所示的完全二叉树的顺序存储如图 5.24 所示。对于列表索引为 i 的结点，其父结点的数组下标为 $\lfloor (i+1)/2 \rfloor -1$，其左右子结点的下标分别为 $2i+1$ 和 $2i+2$。也就是说，用数组来存储完全二叉树和满二叉树，是很容易找到每个结点的父结点和子结点的。

	A	B	C	D	E	F	G	H	I	J	K	L
列表索引值	0	1	2	3	4	5	6	7	8	9	10	11

图 5.24　完全二叉树的顺序存储示意图

2）链式存储结构

对于一般的二叉树，如果仍按从上至下和从左到右的顺序将树中的结点顺序存储在一维数组中，则数组元素下标之间的关系不能够反映二叉树中结点之间的逻辑关系，只有增添一些并不存在的空结点，使之成为一棵完全二叉树的形式，然后再用一维数组顺序存储。这种方式对于需增加许多空结点才能将一棵二叉树改造成为一棵完全二叉树的存储，会造成存储空间的大量浪费。此时可以采用链式存储结构。二叉树的链式存储结构是指用链表来表示一棵二叉树。链表中每个结点由三个域组成，除了数据域外，还有两个指针域，分别用来给出该结点的左子结点和右子结点所在的链结点的存储地址。结点的存储结构为：

lchild	data	rchild

其中，data域存放结点的数据信息；lchild与rchild分别存放指向左子结点和右子结点的指针，当左子结点或右子结点不存在时，相应指针域值为空（用符号∧或null表示）。链式存储结构比较适合存储一般二叉树。

图5.25给出了图5.23中的二叉树的链式存储。

用于表示图5.25的字典定义如下：

```
tree_dic = {'A':('B','C'),'B':('D','none'),'C':('E','F'),'D':('none','G'),
            'E':'leaf','F':('H','none'),'G':'leaf', 'H': 'leaf'}
```

说明：'B':('D','none')表示B结点的左子结点为D、右子结点为空（none），'H': 'leaf'表示结点H为叶结点。

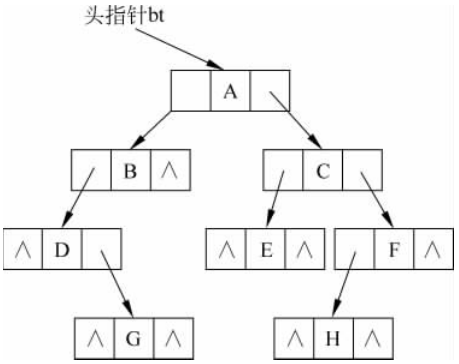

图5.25　二叉树的链式存储

5.3.4　图状结构

图（graph）是一种比树形结构更复杂的非线性结构。在树形结构中，数据元素间具有明显的层次关系，每一层上的数据元素只能和上一层中的至多一个数据元素相关，但可能和下一层的多个数据元素相关。在图状结构中，任意两个数据元素之间都可能相关，即数据元素之间的邻接关系可以是任意的。因此，图状结构被用于描述各种复杂的数据对象，如铁路交通图、通信网络结构、国家之间的外交关系、人与人之间的社会关系等。离散数学是计算机科学的重要数学基础，而图论是离散数学的重要组成部分。

1. 图的定义和术语

图是由非空的顶点集合和一个描述顶点之间关系——边（弧）的集合组成的，其形式化定义为：$G=(V,E)$；其中$V=\{v_i \mid v_i \in \text{dataobject}\}$；$E=\{(v_i,v_j) \mid v_i,v_j \in V \wedge P(v_i,v_j)\}$。$G$表示一个图，$V$是图$G$中顶点的集合，顶点集合构成数据对象（dataobject），顶点就代表数据元素，E是图G中边的集合，集合E中$P(v_i,v_j)$表示顶点v_i和顶点v_j之间有一条直接连线，即偶对(v_i,v_j)表示图中的一条边。图5.26给出了一个图的示例，在该图中，集合$V=\{v_1,v_2,v_3,v_4,v_5\}$；集合$E=\{(v_1,v_2),(v_1,v_4),(v_2,v_3),(v_2,v_5),(v_3,v_4),(v_3,v_5),(v_4,v_5)\}$。

在一个图中，如果任意两个顶点构成的偶对$(v_i,v_j) \in E$是无序的，即顶点之间的连线是没有方向的，称该图为无向图。图5.26中的G_1是一个无向图。在一个图中，如果任意两个顶点构成的偶对$(v_i,v_j) \in E$是有序的，即顶点之间的连线是有方向的，称该图为有向图，如

图 5.27 所示 G_2 是一个有向图。

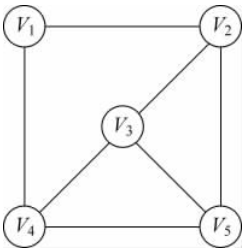

图 5.26　无向图 G_1

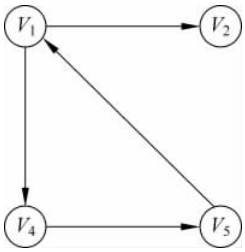

图 5.27　有向图 G_2

在无向图中，顶点之间的连线称为边，用顶点的无序偶对(v_i, v_j)来表示。在有向图中，顶点之间的连线称为弧，用顶点的有序偶对$<v_i, v_j>$来表示。

在一个无向图中，如果任意两个顶点都有一条边直接连接，称该图为无向完全图。在一个有向图中，如果任意两个顶点之间都有方向互为相反的两条弧相连接，称该图为有向完全图。

顶点 v 的度是指连接该顶点的边数，通常记为 TD(v)。在有向图中，要区别顶点的入度与出度的概念。顶点 v 的入度指以该顶点为终点的弧的数目，记为 ID(v)；顶点 v 的出度指以该顶点为始点的弧的数目，记为 OD(v)；有 TD(v)＝ID(v)＋OD(v)。

在无向图中，顶点 v_p 到顶点 v_q 之间的路径(path)是指顶点序列 $v_p, v_{i1}, v_{i2}, \cdots, v_{im}, v_q$。其中，$(v_p, v_{i1}), (v_{i1}, v_{i2}), \cdots, (v_{im}, v_q)$分别为图中的边。路径上边的数目称为路径长度。有向图中的路径定义与此类似。

若一条路径的始点和终点是同一个顶点，则该路径为回路或者环；若路径中的顶点不重复出现，则该路径称为简单路径。

2. 图的存储结构

1) 邻接矩阵

邻接矩阵存储结构，就是用一维数组存储图中顶点的信息，用矩阵表示图中各顶点之间的邻接关系。假设图 $\boldsymbol{G}=(V,E)$有 n 个顶点，即 $V=\{v_0, v_1, \cdots, v_{n-1}\}$，则表示 $\boldsymbol{G}$ 中各顶点相邻关系为一个 $n\times n$ 的矩阵，矩阵的元素为：

$$\mathrm{A[i][j]}=\begin{cases}1 & \text{若}(v_i,v_j)\text{ 或 }<v_i,v_j>\text{ 是图 }\boldsymbol{G}\text{ 的边或弧}\\ 0 & \text{若}(v_i,v_j)\text{ 或 }<v_i,v_j>\text{ 不是图 }\boldsymbol{G}\text{ 的边或弧}\end{cases}$$

图 5.26 的邻接矩阵表示如图 5.28 所示。

2) 邻接表

$$\boldsymbol{A}=\begin{pmatrix}0&1&0&1&0\\1&0&1&0&1\\0&1&0&1&1\\1&0&1&0&1\\0&1&1&1&0\end{pmatrix}$$

图 5.28　无向图 G_1 的邻接矩阵

用邻接矩阵存储图中各顶点之间的关系，有时矩阵非常的稀疏(矩阵中 1 的个数非常少，0 的个数很多)，从而浪费存储空间，此时可以用邻接表存储结构。邻接表是图的一种顺序存储与链式存储结合的存储方法。邻接表表示法类似于树的链表表示法，对于图 $\boldsymbol{G}$ 中的每个顶点 v_i，将所有邻接于 v_i 的顶点 v_j 连成一个单链表，这个单链表就称为顶点 v_i 的邻接表，再将所有顶点的邻接表表头放到数组中，就构成了图的邻接表。在邻接表表示中有两种结点结构，如图 5.29 所示。

一种是顶点表的结点结构，它由顶点域和指向第一条邻接边的指针域构成，另一种是边表

图 5.29　邻接表的结点结构

结点，它由邻接点域和指向下一条邻接边的指针域构成。

图 5.30 给出了无向图 5.26 对应的邻接表表示。

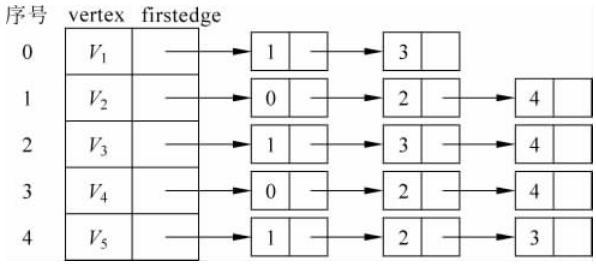

图 5.30　无向图 G_1 的邻接表存储示意图

与图 5.28 对应的列表定义如下：

```
g_list1 = [[0,1,0,1,0],[1,0,1,0,1],[0,1,0,1,1],[1,0,1,0,1],[0,1,1,1,0]]
```

与图 5.30 对应的列表定义如下：

```
g_list2 = [['V1','V2','V4'],['V2','V3','V5'],['V3','V4','V5'],['V4','V5']]
```

5.4　编 译 原 理

用 C、C++或 Python 等高级语言编写程序，相对于早期人们用机器语言编写程序简单、方便得多。但是，计算机并不能直接执行用高级语言编写的源程序，需要翻译成功能等价的机器语言程序才能执行。编译原理就是介绍如何把高级语言源程序翻译成机器语言程序的，学习这方面的知识，既能掌握把高级语言源程序翻译成机器语言程序的基本原理和基本方法，还有助于对高级语言程序设计的深层次理解，提高程序设计能力，培养程序设计思维。

5.4.1　编译程序概述

相对于机器语言，高级语言的优点是简单易学，编写程序和调试修改程序都比较方便，适合于编写规模大、功能复杂的程序。但是，用高级语言编写的源程序需要首先翻译成机器语言程序，计算机才能执行。这种翻译工作如果是手工方式完成，效率非常低而且容易出现翻译错误，很难满足实际需要，完成这种翻译工作的程序称为翻译程序。把高级语言源程序翻译成等价的机器语言程序的翻译程序称为编译程序(compiler)。编译程序也称为编译器。现在的编译程序不仅具有翻译的功能，还有帮助程序员调试修改程序的功能，如指出程序中错误的位置及性质。常说的安装 C 语言程序是在安装 C 语言编译程序（编译器）或叫编译环境，在这个环境下编写的程序叫源程序（source program）。

在计算机上执行一个高级语言源程序一般要分为两步：一是通过编译程序把源程序翻译成机器语言程序，也叫目标程序(object program)，二是执行目标程序。

高级语言源程序的处理也可以采用另一种方式，并不把源程序翻译成机器语言程序然后再执行该机器语言程序，而是采用边翻译边执行的解释执行方式。这种方式下的翻译程序称为解释程序，也叫解释器(interpreter)。

解释方式相对于编译方式效率较低，但解释程序比编译程序容易实现。BASIC 语言和 Python 语言采用的就是解释方式。

不同的编译程序都有自己的组织结构和工作方式，它们都是根据源语言的具体特点和对目标程序的具体要求设计出来的，因此很难给出编译程序的标准结构，也不好说哪种结构好，哪种结构不好。就编译程序所做的工作来看是基本相同的，它主要包括词法分析、语法分析、语义分析、中间代码生成、中间代码优化和目标代码生成等工作。

5.4.2　词法分析

词法分析(lexical analysis)是整个编译过程的第一项工作，其任务就是从左至右逐个字符地对源程序进行扫描，进行词法检查并识别出一个个单词，即通过词法分析把字符序列转换成单词序列，并以机内符的形式表示单词序列，用机内符的形式表示单词是为了便于后续工作的完成。除此之外，词法分析还要完成其他一些相关任务，如过滤掉源程序中的注释和空白，发现词法错误(编写程序时写错关键字等)，指出错误的位置等。

高级语言中的单词一般可以分成 5 类：

(1) 基本字，也称关键字，如 C 语言中的 for、do、if 等。

(2) 标识符，用来表示各种名字的符号串，如变量名、函数名等。

(3) 常数，各种类型的常数，如整数、实数、字符串等。

(4) 运算符，各种算术运算符、关系运算符，如＋、－、＋＋、－－、＜、＞、＜＝、＞＝等。

(5) 界限符，如逗号(,)、分号(;)等。

目前，常用的方式是把每类单词分别表示成一个自动机，词法分析程序据此判断识别出的单词是否正确，如果正确以某种机内符的形式表示，否则提示修改错误。

【例 5.16】 对于 C 语言的如下部分源程序。

```
s = 0;
for (i = 1; i <= 100; i++) s = s + 1;
```

经词法分析处理后，识别成由 23 个单词组成的单词序列，如下：

s	=	0	;	for	(	i	=	1	;	i	<=	100	;	i	++	)	s	=	s	+	1	;

5.4.3　语法分析

语法分析(syntax analysis)的任务是确认作为词法分析结果的单词序列是否为给定语言的一个正确程序。在语法分析中，给定语言用文法表示，如果给定的程序(此时看作单词串或机内符串)能够与该文法相匹配，则认为程序是正确的，否则程序是错误的。单词序列与给定文法匹配的方式有两种：自顶向下分析法和自底向上分析法。

自顶向下分析法也称面向目标的分析方法，也就是从文法的开始符出发试图推导出给定

的单词串，如果能够推导出，则说明单词串是语法正确的程序，否则是错误的程序。自顶向下分析法又可分为确定的和不确定的两种，确定的分析方法需对文法有一定的限制，使得推导过程没有回溯，因而实现方法简单、直观，便于手工构造或自动生成语法分析器，是目前常用的方法之一。不确定的方法即带回溯的分析方法，这种方法实际上是一种穷举的试探方法，效率低、代价高，因而很少使用。

自底向上分析方法是从给定的单词串开始，试图归约（推导的逆操作）为文法的开始符，如果能归约到文法开始符，则确认单词串是正确的程序，否则是错误的程序。与自顶向下语法分析类似。

自顶向下语法分析要解决的问题是如何快速、确定地找到合适的推导过程（如果存在），自底向上语法分析要解决的问题是如何快速、确定地找到合适的归约过程（如果存在）。为此人们提出了一些有效的方法，用于自顶向下分析的方法有递归子程序法、预测分析法等；用于自底向上分析的方法有简单优先法、算符优先法、LR 分析法等。

编译程序的任务是把源程序翻译成目标程序，这个目标程序必须和源程序语义等价，也就是说，尽管它们的语法结构不同，但它们所表达的逻辑含义应完全相同。在词法分析程序和语法分析程序对源程序的语法结构进行分析之后，一般要由语法分析程序调用相应的语义子程序进行语义处理。

编译中的语义处理主要实现的功能是审查每个语法结构的静态语义，即验证语法结构合法的程序是否真正有意义，有时把这个工作称为静态语义分析或静态审查。

5.4.4 中间代码生成

中间代码（middle code）也称中间语言，是复杂性介于源程序语言和机器语言之间的一种表示形式。一般，快速编译程序直接生成目标代码，没有将中间代码翻译成目标代码的额外开销。但是为了使编译程序结构在逻辑上更为简单明确，常采用中间代码，这样可以将与机器相关的某些实现细节置于代码生成阶段仔细处理，并且可以在中间代码一级进行计算和优化工作，使得计算和代码优化比较容易实现。

常用的中间代码形式有逆波兰式、三元式和四元式等。

【例 5.17】 对于表达式 x1＋y2＊6，先变换成 a＋b＊c 的形式。

a＋b＊c 的逆波兰式形式为：abc＊＋。

对于表达式（x1＋y2）＊6，先变换成（a＋b）＊c 的形式。

（a＋b）＊c 的逆波兰式形式为 ab＋c＊。

采用逆波兰式形式表示表达式，可以简化表达式的计算，在从左至右的扫描中，遇到运算符就可以计算，运算对象在该运算符的左边。而在一般表达式中，遇到运算符，需要先比较和其他运算符的优先级别，然后才能决定是否进行计算，对计算机来讲处理难度比较大，效率比较低。

对于逆波兰式 abc＊＋，计算机先扫描到运算对象 a、b 和 c，然后扫描到运算符＊，先计算 b＊c（假定结果为 t），继续扫描到运算符＋，再计算 a＋t，从而完成 a＋b＊c 的计算。无论表达式多复杂，只一遍扫描就能完成表达式的计算。

对于一般表达式 a＋b＊c，计算机先扫描到运算对象 a，然后扫描到运算符＋和运算对象 b，由于不知道后面的运算符是什么，不能决定是否先完成＋的运算，继续扫描到运算符＊和运算对象 c，知道＊的优先级高，先计算 b＊c（假定结果为 t），再往回扫描计算 a＋t。对于比较

复杂的表达式，可能需要多次来回扫描表达式，才能完成计算，这会很浪费时间。

5.4.5 中间代码优化

中间代码优化(middle code optimization)的任务是对中间代码进行等价变换，使得变换后的代码运行结果与变换前代码运行结果相同，而效率提高(运行速度提高或存储空间减少)。

常用的优化技术有删除多余运算、代码外提、强度削弱、变换循环控制条件、合并已知量与复写传播和删除无用赋值等。

【例 5.18】 对于语句 for (i=1; i<=1000; i++)sum=sum+x1+y2*2*3。

要循环执行 1000 次，每次循环做 2 次乘法和 2 次加法，共计 2000 次乘法和 2000 次加法。在循环体中，由于运算对象 x1、y2、2、3 都与循环变量 i 无关，所以在 1000 次循环运算中，x1、y2、2、3 的值是保持不变的，可以提取到循环体外先行计算，2*3 更是可以先计算出结果来。

语句可以改写如下：

```
t = x1 + y2 * 6;
for (i = 1; i <= 1000; i++ )sum = sum + t;
```

上述两个语句合起来，共计需要做 1 次乘法，1001 次加法，大大减少了计算工作量。这种优化工作可以在编写程序时进行，也可以由编译程序在编译时进行，而有些优化工作只能在编译阶段进行。

5.4.6 目标代码生成

目标代码生成(object code generation)是把经过优化后的中间代码作为输入，将其转换成特定机器的机器语言程序或汇编语言程序作为输出，这样的转换程序称为代码生成器，因此，代码生成器的构造与输入的中间代码形式和输出的目标代码的机器结构密切相关。特别是高级语言的多样性和计算机结构的多样性为代码生成的理论研究和实现技术带来很大的复杂性。

由于一个高级语言源程序的目标代码需多次使用，因而代码生成器的设计要着重考虑目标代码的质量问题。衡量目标代码的质量主要从占用空间和执行时间两个方面综合考虑。到底产生什么样的目标代码取决于具体的机器结构、指令格式、字长及寄存器的个数和种类等，并与指令的语义和所用操作系统、存储管理等密切相关。

【例 5.19】 t=abc*+；(表达式为逆波兰式形式的中间代码)

可以生成如下汇编语言程序段：

```
MOV R1,c        /* c 的值送寄存器 R1 */
MUL R1,b        /* R1 中的值与 b 的值相乘并送回寄存器 R1 */
ADD R1,a        /* R1 中的值与 a 的值相加并送回寄存器 R1 */
MOV t,R1        /* R1 寄存器中的值送 t 单元 */
```

5.4.7 编译程序的开发

编译程序是一种相当复杂的系统软件，通常有上万甚至数万条指令。随着编译技术的发展，编译程序的生成周期也在逐渐缩短，但仍然需要很多人年的工作量，而且工作很艰巨，正确性也不易保证。

开发者的愿望是尽可能多地把编译程序的生成工作交给计算机去完成，即编译程序自动

化或自动生成编译程序。计算机科学家们为了实现编译程序的自动生成做了大量的工作，已有词法分析程序的自动生成系统和语法分析程序的自动生成系统。

编译程序自动生成的最关键是语义处理问题，语义处理的自动化与语义描述的形式化工作直接有关。近年来形式语义学的研究取得了很大的进展，大大推动了编译程序的自动生成研究工作，已出现了一些编译程序的自动生成系统，其中比较著名的系统有 GAG、HLP、SIS 和 CGSG 等。形式语义学和编译技术的发展已能实现编译程序的自动生成，目前的主要问题是时间效率、空间节省问题。由自动生成系统产生的编译程序，比人工开发的编译程序长，占用更多的存储空间，运行效率也比较低。

5.5 小　　结

程序设计能力、程序设计思维是计算机专业学生应具备的基本能力和素质，在计算机成为各行各业基本工具的今天，操作使用计算机已不再是计算机专业人员的优势。计算机专业人员要发挥专业特长，在工作中有竞争力，较强的程序设计能力和软件开发能力是坚实的基础。

与提高程序设计能力直接相关的知识有程序设计语言、数据结构、编译原理和算法设计与分析。编写程序，首先要熟悉程序设计语言，熟练掌握基本的程序设计方法和程序调试运行环境。对于数据量比较大或数据之间关系比较复杂的程序，还要熟悉数据结构的知识，选用合适的数据结构合理地组织数据。虽然初学时，程序设计语言的基本知识和基本的程序设计方法的掌握也不是很容易，但计算机专业学生在具备一定的程序设计能力的基础上，重点还是要培养和提高算法设计能力。高级语言是目前常用的程序设计语言，相对来说，高级语言易于学习和掌握，易于编写程序，易于发现和改正程序中的错误，但用高级语言编写的源程序需要翻译成等价的机器语言程序，才能被计算机执行。编译原理就是介绍如何把高级语言源程序翻译成机器语言程序的，学习编译原理知识对于深入理解高级语言程序的执行过程，对于提高较大规模软件的开发能力是非常有益的。

拓展阅读：王选与激光照排

王选(1937—2006)，祖籍江苏无锡，出生于上海市，1958 年在北京大学数学力学系计算数学专业毕业后参加工作。曾任北京大学教授，中国科学院院士，中国工程院院士，2001 年国家最高科学技术奖获得者。

20 世纪 70 年代以前的相当长的时间内，印刷领域广泛使用的是铅字印刷，这和北宋时期毕昇发明的活字印刷没有什么本质的区别，排字工人的工作量非常繁重，排版效率低，印刷质量差，如何利用先进的现代技术改革排版和印刷模式是需要研究解决的重大课题。1975 年，一个偶然的机会，王选听说了国家“汉字信息处理系统工程”研究项目，这个项目于 1974 年 8 月立项，通称“七四八工程”，其中的“汉字精密照排系统”引起了他的浓厚兴趣。如果将计算机技术引入印刷行业，无疑将引起我国报业、出版印刷业等媒体传播领域一场深刻革命，更将对计算机信息技术在我国的普及应用起到推动作用。

当时日本流行的是光学机械式二代照排机，采用机械方式选字，体积大，功能差；欧美流

行的是阴极射线管式三代照排机，对底片灵敏度要求很高，国产底片不容易过关；英国正在研制激光照排四代机，但还没有形成产品。当时国内从事汉字照排系统研究的单位，采用二代机或三代机的模拟存储方法，但由于与西文相比，汉字字形的信息量非常大(英文大小写一共才有 52 个字母，而汉字最少也得使用近 7000 个才能满足最基本的印刷需要)，难以解决存储和输出等技术难题。1976 年王选作出了一个大胆的科学决策：采取跨越式发展的技术路线，跨过第二代和第三代照排系统，直接研制第四代激光照排系统。

选择激光照排方案，需要解决汉字字形信息量太大的难题。王选在 1976 年决定使用“轮廓描述方法”描述汉字字形，为保证字形变大变小时的质量，还提出并实现了用“参数描述方法”控制字形变倍和变形时敏感部位的质量，而西方在大约 10 年后的 80 年代中期才开始采用类似技术。这些方法使字形信息量压缩约 500 倍，达到当时世界最高水平。

把汉字字形信息压缩后存入计算机，还必须将其快速还原和输出。当时小型计算机运算速度很慢，如果用软件实现压缩信息的还原，一秒钟大约只能还原一个汉字。由于王选有多年的硬件实践经验，并懂得微程序，在 1979 年他提出了适合硬件实现的、失真最小的高速还原汉字字形算法，并编写微程序予以实现，使还原速度达到每秒 250 字。后来他又设计出一种加速字形复原的超大规模专用芯片，实现了高速和高保真的汉字字形复原和变倍、变形，使复原速度上升到每秒 710 字，达到当时汉字输出的世界最快速度。

在这些技术创新的基础上，1976—1993 年，王选先后主持设计并实现了六代汉字激光照排控制器，采用双极型微处理器与专用芯片相结合的技术，在计算能力和存储能力较低的计算机系统上完成了页面描述语言的解释处理，使我国的电子出版技术处于世界先进水平，共获得 8 项中国专利，1 项欧洲专利，成为我国第一位欧洲专利获得者。

1991—1994 年，王选率领他的团队不断创新技术，又引发了我国报业和印刷业的三次技术跨越。一是跨过报纸的传真机传版作业方式，直接推广以页面描述语言为基础的远程传版新技术；二是跨过传统的电子分色机阶段，直接研制开放式彩色桌面出版系统；三是规划和组织研制新闻采编流程计算机管理系统，使报社实现网络化生产与管理。这些电子出版新技术的应用，推动和促进了整个印刷行业的技术和设备改造，书刊和新闻出版业呈现空前繁荣。

现在我们都在享受着王选的研究成果，图书、报纸和杂志上的精美图片和文字都是通过激光照排系统印刷上去的。为了纪念王选“自主创新，锲而不舍”的精神，2006 年 10 月，中国计算机学会的“创新奖”正式更名为“王选奖”，每年评选一次。

注：国家最高科学技术奖授予下列科学技术工作者：①在当代科学技术前沿取得重大突破或者在科学技术发展中有卓越建树的；②在科学技术创新、科学技术成果转化和高技术产业化中，创造巨大经济效益或者社会效益的。国家最高科学技术奖每年授予人数不超过 2 名。2000 年开始评选国家最高科学技术奖。国家最高科学技术奖奖金设立之初为每人 500 万元，其中 450 万元由获奖者自主选题，用做科研经费，50 万元属获奖者个人所属。从 2018 年度获奖者开始，奖金调整为每人 800 万元，全部由获奖者个人支配。

习　　题

1. 名词解释：指令、语句、机器语言、汇编语言、高级语言、常量、变量、程序、算法、时间复杂度、空间复杂度、数据、数据项、数据元素、数据对象、数据结构、线性结构、树、森林、二叉树、完全二叉树、满二叉树、图、编译程序、解释程序。

2. 对比说明机器语言、汇编语言和高级语言各自的特点。
3. 结构化程序由哪三种基本结构组成？
4. 良好的程序设计风格主要包括哪些内容？
5. 以列表为例说明数据结构在程序设计中的作用。
6. 说明程序与算法的联系和区别。
7. 简要说明算法的特点。
8. 如何评价算法的优劣？
9. 程序员、高级程序员和系统分析师各自的主要职责分别是什么？
10. 简要说明二叉树的两种存储结构。
11. 简要说明图的两种存储结构。
12. 编译程序由哪几个主要部分组成？每个部分的功能是什么？

思 考 题

1. 程序设计语言、数据结构、算法分析与设计、编译原理4门课程对于提高程序设计能力和培养程序设计思维各有什么作用？
2. 学习一种高级语言，应主要掌握哪些内容？
3. 高级语言中，变量一般要先定义再使用，这样一种规定有什么好处？
4. 如何提高程序设计能力？
5. 查阅文献，举例说明树或二叉树在程序设计中的作用。
6. 查阅文献，举例说明图在程序设计中的作用。
7. 开发编译程序的最难点是什么？