Chapter 3

第 3 章　程序设计语言与程序设计

1. 知识结构

2. 学习目标

- 理解程序语言类型；
- 掌握高级程序语言的基本组成；
- 了解常用高级语言处理程序；
- 理解程序设计范型；
- 掌握程序设计步骤。

3.1　程序设计语言

程序设计语言是为了描述计算过程而设计的一种具有语法语义描述的记号的集合。对计算机使用者而言，程序设计语言是除计算机本身之外的所有工具中最重要的工具，是其他

所有工具的基础。由于程序设计语言的这种重要性，从计算机问世至今的六十多年中，人们一直在研制更新更好的程序设计语言，程序设计语言的数量在不断激增，目前已问世的各种程序设计语言有成百上千个，但这其中只有极少数得到了广泛应用。

程序设计语言是与现代计算机共同诞生、共同发展的，至今已有六十余年的历史，早已形成了规模庞大的系列。随着计算机的日益普及和性能的不断改进，程序设计语言也相应得到了迅猛发展。

程序设计语言发展经历了机器语言时代、汇编语言时代和高级语言发展过程。从机器语言到汇编语言的发展，由于助记符的采用，增强了程序设计语言的可记忆性和提高了源代码的可读性；从汇编语言到高级语言的发展，则增强了程序设计语言的可理解性和提高了语言表达的效率。程序设计语言的分类如图 3-1 所示。

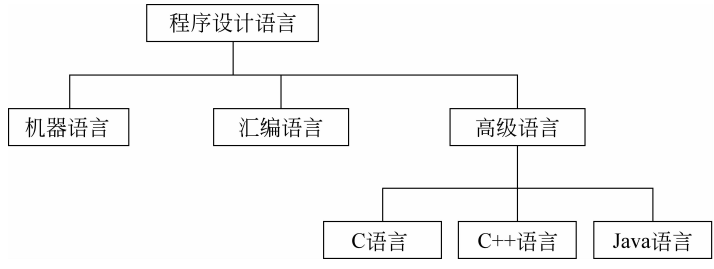
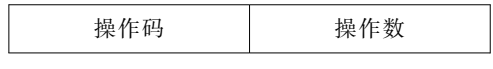


图 3-1　程序设计语言的分类

3.1.1　机器语言

最先使用的计算机程序设计语言是机器语言。机器语言的所有指令都是由 0 和 1 组成的二进制数。计算机发明之初，人们只能用计算机的语言去命令计算机，即写出一串串由 0 和 1 组成的指令序列交由计算机执行，这种语言就是机器语言，也就是第一代计算机语言，它是计算机能直接理解和执行的语言，机器语言中每一条指令实际上是一条二进制形式的指令代码，指令格式如下：



操作码指出应该进行什么样的操作，操作数指出参与操作的数本身或它所在的地址。

例如：计算 $A=15+10$ 的机器语言程序如下。

```
10110000  00001111      ：把 15 放入累加器 A 中
00101100  00001010      ：10 与累加器 A 中的值相加，结果仍放入 A 中
11110100                ：结束，停机
```

由此可以看出，机器指令的功能很弱，而且记忆困难，很多工作（如把十进制数表示为计算机能识别的二进制数）都要人工完成。因此，用机器语言书写程序时，程序设计人员不仅非常费力，而且编写程序的效率还非常低。使用机器语言十分困难，特别是在程序有错需要修改时，更是如此。而且，由于每台计算机的指令系统往往各不相同，所以，在一台计算机上执行的程序，要想在另一台计算机上执行，必须另外设计程序，造成了重复工作。但由于使

用的是针对特定型号计算机的语言,因此机器语言程序的执行速度快。

3.1.2 汇编语言

为了增强机器语言程序的可理解性和可阅读性,用一些简洁的英文字母、符号串等助记符来代替一个特定指令的二进制串,即用助记符来代替指令的操作码和地址码。例如,用ADD代表加法,MOV代表数据传输等,这样增加了可理解性、可维护性,通常将这种程序设计语言称为汇编语言。利用汇编语言编写的程序又称为汇编语言程序。

例如,计算 $A=15+10$ 的汇编语言程序如下。

```
MOV     A,15        ;把 15 放入累加器 A 中
ADD     A,10        ;10 与累加器 A 中的值相加,结果仍放入 A 中
HLT                 ;结束,停机
```

汇编语言也是一种面向机器的程序设计语言,它与机器的逻辑结构相关,用助记符号来表示机器指令的操作码与操作数(即运算符与运算对象)。然而计算机是不认识这些符号的,这就需要一个专门的程序,专门负责将这些汇编语言程序翻译成计算机能够直接理解与执行的机器语言程序,负责翻译的程序又被称为汇编程序。汇编程序设计序就是完成这种转换工作的一种专门的程序。汇编程序设计是把用汇编语言编写的程序(即源程序)翻译为等价的机器语言程序(即目标程序)的一种程序。汇编语言的出现使人们在编写程序时不必再花较多的精力去记忆、查询机器代码与地址,程序设计工作变得更为容易。

机器语言中用机器指令来表示机器语言中某个特殊的操作。类似地,汇编语言中用汇编指令来表示汇编语言中某个特殊的操作。汇编语言和机器语言基本上是一一对应的。也就是说,对大多数汇编语言中的指令来说,在机器语言中都存在一条功能相同的机器指令。例如:假设汇编语言中用LOAD表示取数操作,对应机器指令的操作码为10;汇编语言中用STORE表示存数操作,对应机器指令的操作码为20;汇编语言中用ADD表示加法操作,对应机器指令的操作码为30;汇编语言中用HALT表示结束程序运行操作,对应机器指令的操作码为00等。

尽管与机器语言相比,汇编语言的抽象程度要高得多,但由于它们之间是一对一的关系,用它编写一个很简单的程序,也要使用数百条指令。为了解决这个问题,研制出了宏汇编语言,一条宏汇编指令可以翻译成多条机器指令,这使得人们的程序设计工作量得以减轻。为了解决由多人编写的大程序的拼装问题,研制出了连接程序,它用于把多个独立编写的程序块连接组装成一个完整的程序。连接过程如图3-2所示。

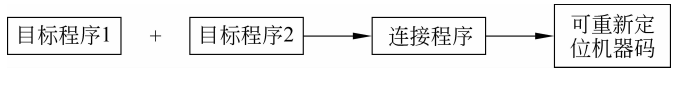


图 3-2 连接程序

汇编语言同样十分依赖于机器硬件,移植性不好,但效率仍十分高、速度快,针对计算机特定硬件而编制的汇编语言程序,能准确发挥计算机硬件的功能和特长,程序精炼而质量高,所以至今仍是一种常用而强有力的软件开发语言,目前大多数外部设备的驱动程序都是

用汇编语言编写的。

3.1.3　高级语言

1. 高级语言简介

虽然用汇编语言编写程序比用机器语言编写程序方便，但用汇编语言编写程序仍然不是一件容易的事情。从最初与计算机交流的经历中，人们意识到应该设计一种接近于数学语言或人类的自然语言，同时又不依赖于具体的计算机结构，编出的程序能在所有机器上通用。这种语言就是高级程序设计语言(简称高级语言)，即第三代计算机语言。高级语言是在伪码形式描述算法的基础上发展而来的，与汇编语言相比，高级程序设计语言的抽象度高，与具体计算机的相关度低(或没有相关度)，容易理解和容易描述解题过程的程序语言，求解问题的方法描述直观，因此用高级语言设计程序的难度较以前大大降低。

世界上第一个高级程序设计语言是在 20 世纪 50 年代由 John Backus 领导的一个小组研制的 FORTRAN 语言。Backus 当时一边研制 FORTRAN，一边研究了将这种代数语言翻译成机器语言的可能性。由于人们当时头脑中最关心的问题不是这种语言能否设计与编译，而是担心它经翻译后执行时能不能保持一定的运行效率。这一担心大大影响了 FORTRAN 的设计方向。最早的一个 FORTRAN 版本 FORTRAN 0 是 1954 年前后设计出来的，其编译程序于两年后研制出来。与此同时，其后继版本 FORTRAN Ⅰ与 FORTRAN Ⅱ也相继问世。FORTRAN 一问世便受到了极大的欢迎并很快流行起来。FORTRAN 是一个处处注重效率的语言，这从其各种成分(如控制语句中表达式与数组上标中的表达式的形式以及子程序参数的传送方式等)就可以看出来。FORTRAN 首先引入了与汇编语言中的助记符有本质区别的变量的概念，奠定了程序设计语言中名字理论的基础。它所引入的表达式、语句、子程序等概念也是高级程序设计语言的重要基石。但由于 FORTRAN 是从低级语言发展而来的，其许多成分中“低级”的痕迹随处可见。目前常用的 FORTRAN 版本是 FORTRAN 77 与 FORTRAN 90。

随着 FORTRAN 语言的成功和不断发展，又有许多高级程序设计语言被提出，如 ALGOL 语言、COBOL 语言等。在程序设计语言几十年的发展历程中，推出的高级语言不下百种。有重要意义的有几十种，影响较大、使用较普遍的有 FORTRAN、ALGOL、COBOL、BASIC、LISP、SNOBOL、PL 1、PASCAL、C、PROLOG、Ada、C ++ 、VC、VB、Delphi、Java 等。随着程序设计语言的不断发展，许多高级语言由于先天不足，或后天没有软件商对其进行持续的更新和改造，逐渐被市场淘汰，如 ALGOL 语言目前就不再被使用。

高级语言的出现推动了软件的发展，也是目前计算机得到广泛应用的一个重要原因。

2. 高级语言的发展

高级语言的发展经历了从早期的语言到结构化程序设计语言、从面向过程的语言到非过程化程序语言的过程。相应地，软件的开发也由最初的个体手工作坊式的封闭式生产，发展为产业化、流水线式的工业化生产。

1) 高级语言初创时期

高级程序语言的初创时期主要是在 20 世纪 50 年代，主要的代表语言有 FORTRAN 语言、ALGOL 语言、COBOL 语言。

2) 高级语言发展初期

在20世纪60年代初期,编译技术与理论的研究得到了高度重视,在短短几年中得到了很大发展,许多语言翻译中的问题也得到了解决,这又反过来使人们把注意力放在各种新的程序设计语言的研制上,这就导致了程序设计语言数目指数般地激增。在20世纪60年代的10年中,人们至少研制了200种高级语言。其中较为成功的有LISP语言、BASIC语言等。

3) 结构程序设计时期

1968年E. W. Dijkstra给*COMM. ACM*杂志编辑写了一封信,指出了语言中转向语句使用上带来的问题,从而引发了程序设计语言中要不要使用转向语句的讨论,这场讨论使人们开始注重对程序设计方法进行研究,从而导致了结构程序设计这一新的程序设计方法问世。这一时期比较著名的语言有PASCAL、Modula、C、Ada等。

4) 多范型程序设计语言时期

在高级程序设计语言问世以后的几十年内,尽管在20世纪60年代问世了LISP、APL与SNOBOL4等非过程式(非强制式)程序设计语言,但仍然是过程性语言的天下。但自从J. Backus在1978年图灵奖获奖讲演中指出了传统过程性语言的不足之后,人们开始把注意力转向研究其他风格、其他范型的程序设计语言。目前已被人们研究或应用的非过程式语言范型主要有:函数式语言、逻辑式语言、面向对象式语言与关系式语言等几种。

3. 高级语言的分类

高级语言常用的分类方法有基于设计要求划分、基于应用范围划分、基于描述问题的方式划分、按照使用方式划分及按照成分性质划分等。其中,基于描述问题的方式对高级语言进行分类是最常用的分类方法。

(1) 基于设计要求划分。

按照用户的要求,程序设计语言分为过程式语言和非过程式语言。过程式语言的主要特征是,用户可以指明一列可顺序执行的运算,以表示相应的计算过程,如FORTRAN、COBOL、PASCAL等。非过程式语言的定义是相对于过程式语言来说的,凡是设计者无法表示出求解过程的一列可以顺序执行的运算步骤的语言都是非过程式语言。非过程式语言的典型代表有PROLOG语言。例如,用PROLOG语言编写的程序以逻辑推理为问题求解的基础,而不是通过给出一列可以顺序执行的运算步骤来描述求解步骤的。PROLOG语言程序的执行过程是按照程序语句的逻辑次序来执行的,这种逻辑次序和FORTRAN语言描述的执行过程是完全不相同的。

(2) 基于描述问题的方式分。

可把高级语言分成命令型语言、函数型语言、描述型语言和面向对象型语言。命令型语言是出现最早和曾经使用最多的高级语言。命令型语言的特点是计算机按照该语言描述的操作步骤来执行。换句话说,命令型语言程序中的语句就是要求计算机执行的命令。FORTRAN语言、COBOL语言、ALGOL语言、BASIC语言、C语言、PASCAL语言、Ada语言、APL语言等都属于命令型语言。函数型语言的特点是把问题求解过程表示成"块"结构,对调用"块"的调用者来说,每个"块"都有输入数据和经过加工处理后的输出数据。这样,每个"块"的功能就像数学家所说的"函数"的功能,所以这种语言称为函数型语言。

LISP 语言、ML 语言等属于函数型语言。如果说命令型语言强调的是求解问题的步骤是什么的话，那么，描述型语言强调的是问题是什么。描述型语言的特点是设计者给出的是问题的描述，计算机根据问题描述的逻辑进行处理。由于这类高级语言是基于逻辑的，所以也称为逻辑型语言。PROLOG 语言、GPSS 语言等属于描述型语言。面向对象型语言的特点是把数据以及处理它们的子程序统一作为对象封装在一起进行处理。

(3) 基于应用范围分类。

有通用语言与专用语言之分。目标单一的语言称为专用语言，如 APT 等；非目标专一的语言称为通用语言，如 FORTRAN、COLBAL、PASCAL、C 等都是通用语言。

(4) 按照使用方式，有交互式语言和非交互式语言之分。具有反映人机交互作用的语言称为交互式语言，如 BASIC 等；不反映人机交互作用的语言称为非交互式语言，如 FORTRAN、COBOL、ALGOL69、PASCAL、C 等都是非交互式语言。

(5) 按照成分性质，有顺序语言、并发语言和分布语言之分。只含顺序成分的语言称为顺序语言，如 FORTRAN、C 等；含有并发成分的语言称为并发语言，如 PASCAL、Modula 和 Ada 等。

程序设计语言是软件的重要方面，其发展趋势是模块化、简明化、形式化、并行化和可视化。由于以对象为基础的面向对象的高级语言较传统程序设计语言更符合人类思维和求解问题的方式，所以近年来，面向对象的高级语言有了长足的发展。面向对象的高级语言是目前程序设计语言发展的主流方向。

3.2　高级程序语言的基本构成

程序设计语言是人们为了描述计算过程而设计的一种具有语法语义描述的记号，是人与计算机进行交往的工具 ，它本身作为语言自然存在大量的词法、语法、语义等多种约定，本节将介绍高级程序设计语言的基本构成，并以 C 语言为例进行说明。

3.2.1　变量、运算符和表达式

1. 变量

在程序中都以变量的方式存储参与计算的数据、计算的中间结果和最后结果。但通常都要求变量在使用前必须先定义，声明其类型和名称，然后翻译程序。根据变量的数据类型为该变量分配相应的存储空间，从而存储该变量的值。从高级语言的角度看，变量代表了特定大小的内存单元空间。

高级语言把变量按作用域分为全局变量和局部变量。全局变量是允许所有模块都使用的变量，常用来存储公共数据。局部变量是只允许在一个过程体内使用的变量。

C 语言中变量声明的方法如下：

```
int value=3;
double d=5.5;
```

上面分别声明了一个整型、双精度型的变量。其中 value、d 为变量的名字;3、5.5 为变量的值。由上面可见声明一个变量的方法为

<类型><变量名>;

2. 运算符

计算机既可以进行加、减、乘、除等算术运算,也可以进行与、或、非等逻辑运算,这些都是通过指明运算符实现的。高级程序设计语言提供的运算符的种类和运算符的表示方式有所差异,分类方式也会有所不同。

按操作数的数目来分,可以有一元运算符、二元运算符(如+、>)和三元运算符,它们分别对应于一个、两个和三个操作数。此处操作数可以理解为运算符作用的变量或对象。按照运算符功能来分,基本的运算符有下面几类:

(1) 算术运算符(加+、减-、乘*、除/等);

(2) 关系运算符(大于>、小于<、、等于==、不等于!=等);

(3) 布尔逻辑运算符(与 &&、或||、非! 等);

(4) 赋值运算符(=及其扩展赋值运算符);

(5) 条件运算符(?:)。

3. 表达式

表达式是由常量、变量、函数、运算符和括号组成的有意义的式子,其目的是用来说明一个计算过程。例如:c=a+b、d=a*b 就是两个简单的表达式。

3.2.2 数据类型

由于不同类型的数值占用内存单元的大小不同,所以高级语言在进行变量定义时,要具体指出该变量要存放数值的类型。高级语言中引入数据类型的概念来解决这一问题。在高级语言中,定义变量就是指明该变量的数据类型,从而为该变量分配相应数据类型的内存空间。

例如:C 语言规定整数有三种数据类型,分别是长整数、整数和短整数,长整数用符号 long(或 long int)表示,整数用符号 int 表示,短整数用符号 short(或 short int)表示。其中,int 数据类型占用 2B,由于 2B 可表示的数据范围是-32 768~+32 767,因此定义为 int 数据类型变量的数据范围是-32 768~+32 767。当定义为 int 数据类型的变量超出了这个范围,数据就会出错,其程序也就必然会出错。long 数据类型占用 4B,因此 long 数据类型的变量比 int 数据类型的变量可表示的数据范围大。

在高级语言中,数据类型通常分为基本数据类型和构造数据类型两种。

1. 基本数据类型

一般来说,高级语言的数据类型包括整数数据类型、实数数据类型和字符数据类型三种。为节省内存单元空间,整数数据类型又进一步细分为长整数、整数和短整数三种子类型,而实数数据类型又分为单精度和双精度两种子类型。后面还要讨论构造结构的数据类型。由于构造结构的数据类型是由这里讨论的简单数据类型构成的,所以为区别起见,也称这里讨论的简单数据类型为基本数据类型。

为有效和正确地进行程序设计，程序设计人员在定义变量时，首先要根据问题分清该变量应为整数、实数和字符数据类型中的哪一种，然后还要根据问题分清该变量应是哪种子数据类型。如果把所有变量都按占最大内存单元空间的子数据类型来定义，当然程序运行时不会出错，但是浪费的内存资源很多；如果把所有变量都按占最小内存单元空间的子数据类型来定义，内存资源会很节省，但是一旦数据超过该数据类型所允许的数据范围，则程序运行时就会出错。所以定义变量时，数据类型特别是子数据类型的选择要合适。

在高级语言中，数据类型不仅用来确定数据（或变量）所要占用的内存单元大小，还用来进行操作的匹配性检查。例如，假设有以下 C 语言语句：

```
int n;
n=5.5
```

上述程序段中的一行定义了一个 int 类型的变量，第二行语句是给变量 n 赋一个单精度的实数数值，而一个只有 2B 的内存单元的整数类型变量中，是无法存放需要 4B 内存单元的单精度实数数值的。因此系统需要在运行该程序前对该语句提出可能存在错误或可能引起错误的警告，并在实际运行时，只把单精度实数数值 5.5 的整数部分赋值给变量 n。

总结上述讨论，可以得出以下结论：

（1）数据类型是高级语言为变量分配具体大小的内存单元的需要；

（2）数据类型是系统用来检查高级语言程序中表达式计算或变量赋值等是否匹配的需要。

2. 构造数据类型

现实世界中有很多事物的属性是密切相关的，因此程序设计中反映这些事物属性的数据也应该密切相关。例如：描述学生的数据有学号、姓名、性别、年龄等，一个学生的这些数据应该组成一个整体。构造数据类型为程序设计人员提供了把现实世界中有密切联系的数据组织成一个整体的工具。

构造数据类型是在基本数据类型的基础上构造出来的数据类型。数组和结构是大多数高级语言都支持的两种最主要的构造数据类型。

1）数组

数组是相同数据类型的集合。例如，在 C 语言中，假设要使用 10 个 int 型的变量，可以定义如下：

```
int v1,v2,v3,v4,v5,v6,v7,v8,v9,v10;
```

为和数组类型的变量区别，称这里定义的变量为简单变量。由于上述 10 个变量的类型相同，所以也可以把这 10 个变量定义成一个数组。不同的高级语言定义数组和表示数组元素的方法略有不同，但基本原理都相同。在定义变量时，C 语言用变量名后加符号“[ ]”来表示所定义的变量为数组类型的变量，定义如下：

```
int v[10]
```

上述语句定义了包含 10 个变量的数组类型变量。数组变量中的一个变量称做一个数

组元素。C语言规定数组元素的序号从0开始，所以10个数组元素分别是$v[0]$，$v[1]$，$v[2]$，$v[3]$，$v[4]$，$v[5]$，$v[6]$，$v[7]$，$v[8]$，$v[9]$。由于一个数组变量包含了若干个数组元素，所以高级语言在实现时，是给每个数组变量分配地址连续的内存单元块。每个数组元素占2个字节，所以总计占用了20个字节。

由于一个数组变量是由若干个数组元素组成的一个整体，所以数组类型既可以简化变量的定义，还可以方便程序设计。例如，如果上述10个数组元素的数组变量中保存了要累加的数值，在编写循环形式的累加时，就可以把循环体写成sum＝sum＋$v[i]$，其中数组下标i从0增加到9。

2）结构体

结构体是不同数据类型的集合。结构体用来表示一个有若干个分量的事物。例如，在登记一个学生的信息时，需要登记学生的姓名、年龄、平均成绩等信息。希望把描述一个学生各个属性的信息表示成一个有逻辑联系的整体，这样既可以简化变量的定义，也可以使程序更容易看懂。结构体就可以达到这个目的。

不同的高级语言表示结构体的方法不同。在C语言中，上述学生结构体可以定义如下：

```
struct student
{
    char name[8];
    int age;
    float average;
};
```

上述语句就是定义了一个结构体。结构体定义是根据问题的需要，利用基本数据类型定义一种新的数据类型。其中，student为这个结构体的名字，name、age和average是该结构体的三个分量。在上述定义之后，就可以像用数据类型int定义变量一样，用结构体student来定义结构体变量。例如结构体变量可以定义如下：

```
student  s;
```

由于一个结构体变量包含了若干个分量，所以高级语言在实现时，给每个结构体变量分配地址连续的内存单元块。结构体变量s的内存单元分配示意图如图3-3所示。由于char数据类型占1B，name分量为char类型的8个元素的数组，共计占用了8B；int数据类型占2B，所以age分量占2B；float数据类型占4B，所以average分量占4B。因此，结构体变量s总计占用了14B。

name	age	average
8B	2B	4B

图3-3　结构体变量s的内存单元分配

在上述结构体变量s定义之后，就可以用以下三个赋值语句给该变量的三个分量分别赋值：

```
s.name="张三";
s.age=22;
s.average=89.9;
```

3.2.3　赋值语句

变量代表了特定大小的内存单元空间。在定义变量时，只是给变量分配了需要大小的内存单元空间，但这个内存单元中并没有存放任何数值。赋值语句用来实现给变量赋数值。为了人们的使用习惯保持一致，大多数高级语言的赋值语句都用赋值号“＝”表示。赋值号“＝”右边是要赋予的数值，称为右值；“＝”左边是要接受数值的变量，称为左值。C 语言中的赋值语句如下所示：

```
int n;
n=22;
n=n+1;
```

其中，第一条语句定义变量 *n* 为 int 类型的变量，第二条语句给变量 *n* 赋予数值 22，第三条语句修改变量 *n* 中的数值为 23。

变量出现在左值位置和出现在右值位置表示不同的含义：出现在左值位置的变量是要接受赋值数据的变量，所以出现在左值位置的变量表示该变量对应的内存单元地址；出现在右值位置的变量是要赋值的数据，所以出现在右值位置的变量表示该变量对应内存单元中的数值。

3.2.4　输入输出

完整的程序都应该含有输入和输出的功能。没有输出功能的程序是没有用的，无法看到运算结果；没有输入功能的程序缺乏灵活性，每次运行都只能对相同的数据进行加工。所以输入输出是程序中不可缺少的部分。

程序一般都是先输入数据，然后进行相应的数据运算，最后将结果输出。若是在非简单计算的情况下，还能在运行期间获得用户的输入。

程序的输入输出方式有两种：一种是以文件方式进行的程序之间的数据传递；另一种是人-机交互方式，该方式的输入是把数据按一定的格式输入变量中，输出则是按用户要求的格式显示或打印变量的值。该方式的输入、输出在不同的程序设计语言中使用不同的语句或函数来实现。

C 语言库函数中提供了一组输入、输出函数，其中 scanf 和 printf 函数是针对标准输入、输出设备（键盘和显示器）进行格式化输入输出的函数。

例 3-1　用 C 语言实现从键盘上输入一个字符，并将其输出。

```
main()
{
    char c;
    printf("请输入一个字符:");
    scanf("%c",&c);
    printf("您输入的字符为:%c",c);
}
```

程序执行结果：

```
请输入一个字符:f(按 Enter 键)
您输入的字符为:f
```

3.2.5 控制结构

通常，结构化的程序设计包括顺序结构、选择结构和循环结构三种，它构成了程序的主题。各种程序设计语言一般都有这三种结构，但它们的表示形式有所差异。

1. 顺序结构

顺序结构是依据语句出现的先后次序依次执行的，如图 3-4 所示。程序执行了 A 语句再执行 B 语句。由赋值符号构成的赋值语句是典型的顺序结构。

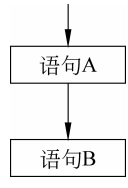


图 3-4 顺序结构

下面是一个用 C 语言实现的顺序结构的程序。

例 3-2 从键盘上输入任意一个大写英文字符，将其转换为小写字母后输出。

```
main()
{
    Char c;
    printf("请输入一个大写英文字母: ");
    scanf("%c",&c);
    c=c+32;
    printf("转换后的小写字母为: %c",c);
}
```

执行过程：

```
请输入一个大写英文字母:B(按 Enter 键)
转换后的小写字母为:b
```

2. 选择结构

选择结构是根据条件(某些变量或表达式的值)判断结果，决定哪些程序段被执行以及哪些程序段不被执行的结构形式。最基本的选择结构语句是 if-then-else 或其变形(if-then)语句，其图示为图 3-5。

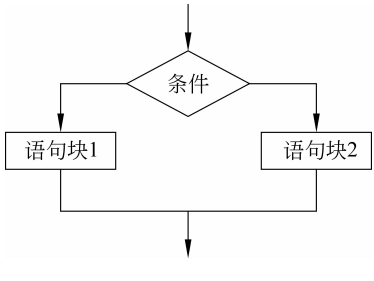


图 3-5 选择结构

不同的高级语言表示 if 语句的格式有所不同。C 语言表示 if-then-else(或其变形 if-then)语句的格式为

```
if(condition)  S1;
else  S2;
```

C 语言规定，一条语句结束时用分号(;)标识，这样 C 语言就把条件语句分成了两条语句。如果只有第一条语句，则表示当条件 condition 为真

时，执行语句组 S1；如果两条语句都有，则表示当条件 condition 为真时，执行语句组 S1，否则执行语句组 S2。这样就方便地实现了 if-then-else 语句及其变形语句 if-then 的功能。当语句组 S1 或 S2 为若干条语句时，需要明确地表示出语句组 S1 或语句组 S2 包含的语句。在 C 语言中，用一对花括号({})括起来的部分表示对应的语句组。

```
if(n<100)
{
    x=x+n;
    n=n+2;
}
else
{
    x=x-n;
    n=n-2;
}
```

上述语句表示，当 $n<100$ 时，执行语句 $x=x+n$ 和语句 $n=n+2$；否则，执行语 $x=x-n$ 和语句 $n=n-2$。有些高级语言(如 PASCAL 语言)用标识符 begin 代表语句组的开始，用标识符 end 代表语句组的结束。

当分支多于两个时，虽然也可以用分支语句嵌套来实现，但是比较麻烦。特别是当分支有三个以上时，这样的表示形式显得很烦琐。对于这种多分支的情况，大多数高级语言提供了一种称为 case 的语句，可以方便地实现多分支情况下的程序设计。在 C 语言中，case 语句的格式为

```
switch(N)
{
    case C1: S1;
    case C2: S2;
    …
    case Cn: Sn:
    default: S0;
}
```

上述语句的语义是：变量 N 可以有多个取值选择，当 N 等于 C1 时，执行语句组 S1；当 N 等于 C2 时，执行语句组 S2；当 N 等于 Cn 时，执行语句组 Sn；当 N 不等于上述任何一个数值时，执行语句组 S0。

例 3-3　现假设要计算函数 $y(x)$的值，其函数表达式如下：

$$y=\begin{cases}100, & x>100\\ 50, & 50<x\leqslant 100\\ 0, & x\leqslant 50\end{cases}$$

用 C 语言的选择结构语句实现上述函数的程序为

```
main()
{
    int x,y;
```

```
    printf("请输入 X 的值: ");
    scanf("%d",&x);
    if(x>100)
        y=100;
    else if(x>50 && x<=100)
        y=50;
    else y=0;
    printf("输入 x 的值%d 对应的 y 为: %d",x,y);
}
```

程序的执行过程如下:

```
请输入 X 的值:230(按 Enter 键)
输入 x 的值 230 对应的 y 为:100
```

例 3-4 现假设要实现以下功能:用户从 A、B、C、D 4 个选项中任意选择一个,根据用户的选择输出用户选择了哪一项及此项代表的含义,如果输入其他字母,则输出无此选项。设 A 代表 Apple,B 代表 Boy,C 代表 Cat,D 代表 Dog,则用 C 语言的 switch 开关语句实现的程序如下:

```
main()
{
    char c;
    printf("请选择 ABCD 中的一项: ");
    scanf("%c",&c);
    switch(c)
    {
        case 'A': printf("选择了 A,代表 Apple\n");
                  break;
        case 'B': printf("选择了 B,代表 Boy\n");
                  break;
        case 'C': printf("选择了 C,代表 Cat\n");
                  break;
        case 'D': printf("选择了 D,代表 Dog\n");
                  break;
        default: printf("输入选项错误,无此选项\n");
    }
}
```

程序执行过程:

```
请选择 ABCD 中的一项:D(按 Enter 键)
选择了 D,代表 Dog
```

3. 循环结构

循环通常有两种情况:一种情况是循环次数不固定,需要根据当前的条件判断来决定是否继续执行循环体;另一种情况是循环次数固定,执行循环体的次数可以明确给出。大部

分高级语言都给出了可以方便地表示这两种情况的循环语句。例如 C 语言对第一种情况的循环语句格式为

```
while(condition)S;
```

该语句的语义是：当条件 condition 为真时，执行语句组 S；当条 condition 为假时，执行该语句后面的语句，如图 3-6 所示。

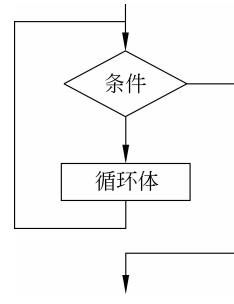


图 3-6　while 循环结构

C 语言对第二种情况的循环语句格式为

```
for(count=nl;cout<=n2; count=count+c)  S;
```

该语句的语义是初始时，计数变量 count 等于数值 $n1$；当计数变量 count 小于数值 $n2$ 时，首先执行语句组 S，然后把计数变量 count 修改为 count 加上某个常数 c 的和；然后继续测试计数变量 count 是否小于数值 $n2$。这样，循环体的语句组 S 将总共被执行 $(n2-n1+1)/c$ 次。

例 3-5　打印 200 以内的所有偶数。

```
main()
{
    int i=0;
    //用 while 语句的实现形式如下
    printf("用 while 语句输出的偶数:\n");
    while(i<=200)
    {
        printf("  %d  ",i);
        i=i+2;
    }
    //用 for 语句的实现形式如下
    printf("用 for 语句输出的偶数:\n");
    for(int j=0; j<=200; j=j+2)
    {
        printf("  %d  ",j);
    }
}
```

程序执行过程如下。

```
用 while 语句输出的偶数:
0 2 4 6 8 ……198 200
用 for 语句输出的偶数:
0 2 4 6 8 ……198 200
```

3.2.6　过程

过程是算法的基本元素，过程可以使一个求解大问题的算法分解为若干个求解子问题

算法的有机合成。许多高级程序设计语言也把过程称做函数。高级语言在实现过程时,要考虑过程的调用、过程的参数、过程的返回、参数的传递方式等问题。

1. 过程的调用

过程主要有两个用途:①构造通用的算法模块;②把一个大的、复杂的算法分解为若干个小的、简单的算法的合成。过程的上述两种用途都涉及一个程序(或过程)对一个过程的调用。

所谓过程的调用,是指停止执行当前的程序,而转去执行被调用的过程,过程的调用如图 3-7 所示。在图 3-7 中,主程序表示当前正在运行的程序,在主程序的运行过程中,如果遇到调用过程的语句,则停止运行当前的主程序,而转去运行被调用的过程,过程运行完后,再返回主程序调用过程语句的下一条语句继续运行。

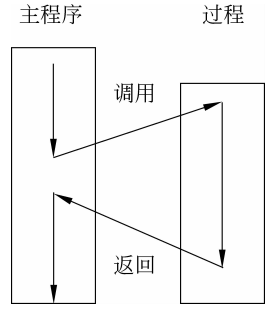


图 3-7 过程调用

2. 过程的参数

通常过程都包含参数,过程的参数使过程的设计具有灵活性和通用性。例如,求 1～n 的累加和问题的 C 语言函数(C 语言把过程称做函数)如下:

```
int Suml(int n)
{
    int i, sum;                    //变量定义
    sum=0;
    for(i=1; i<=n; i=i+1)          //累加初始化
      sum=sum+i;                   //循环累加
    return sum;                    //返回语句
}
```

上述 C 语言函数可以求 1～n 的累加和问题,其中,n 可以是任意的整数数值。这样,高级语言过程的参数就必须是变量。

该变量的具体取值由调用该过程的主程序给出。主程序通过传送具体的数值给被调用过程的参数,来实现具体的过程调用。

过程的引入就使程序可以由多个模块组成,这些模块在具体设计时可能由多个不同的人来完成。为了防止一个过程内的变量数值被起了相同变量名字的其他过程混用,高级语言把变量分为全局变量和局部变量。全局变量是允许所有模块都使用的变量,局部变量是只允许在一个过程体内使用的变量。过程的参数是局部变量。例如,例 3-5 中的参数 n 就是局部变量。

3. 过程的返回

从图 3-7 可知,过程的调用包含调用和返回两个步骤。调用步骤要求主程序传送具体数值给过程,这通过过程的参数来实现。返回步骤要求把过程的计算结果传送给主程序。

由于变量分为局部变量和全局变量,为避免一个过程被外部的程序非法修改,高级语言规定:在一个过程内定义的变量都是局部变量。例如,例 3-5 的过程内定义的变量 sum 就是局部变量。外部主程序是无法得到过程内局部变量的数值的,所以要考虑把过程计算结

果传送给主程序的方法。这样的处理方法主要有两种。这里讨论第一种方法。第二种方法将在讨论参数的地址传送方式时介绍。

高级语言处理过程返回的第一种方法是，把过程名既看作过程名，也看作一个变量。这样首先把过程名这个变量定义成全局变量，然后把过程的计算结果传送给过程名，在主程序中就可以得到计算结果。

不同的高级语言实现把过程的计算结果传送给过程名的方法不同。在 C 语言中，设计了一个专门的语句用来实现把过程的计算结果传送给过程名，该语句的格式如下：

```
return x;
```

其中，return 是该语句的标识符，x 表示要传送给过程名的计算结果变量名。例如，例 3-5 中过程的最后一条语句 return sum；就是用来实现把计算结果 sum 中的数值传送给过程 Sum1 的。

4. 参数的传递方式

主程序通过给被调用过程的参数传送具体的数值来实现过程的调用。主程序和被调用过程之间通过参数传送数据有两种方法。

通过参数传送数据的第一种方法称为值传送。值传送，就是把主程序调用过程的具体数值(称为实际参数，或简称实参)复制给过程的参数(称为形式参数，或简称形参)。图 3-8(a)就是值传送方式实现方法的示意。其中，实参和形参都是内存单元，在过程调用时，把实参中的数值复制给形参。

通过参数传送数据的第二种方法称为地址传送。地址传送，就是把形参表示成内存单元地址形式，此时形参指向的内存单元和实参指向的内存单元相同，或者说，形参和实参共享一个内存单元。图 3-8(b)就是地址传送方式实现方法的示意。

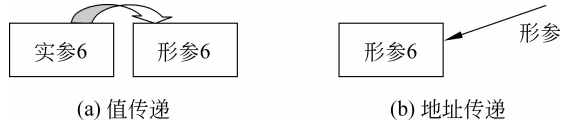


图 3-8　参数传递的方式

参数的传递方式对程序实现的方法有一定的影响。对于值传送来说，由于数据的传送是单方向的，即只有从主程序到过程一个方向，所以运行过程时，参数不会影响主程序中原来的数值。如图 3-9 所示，过程调用时，实参复制数值 6 给形参，若过程运行时形参的数值改变为 8，则过程结束后，主程序中实参的数值还是原来的 6，而不是 8。

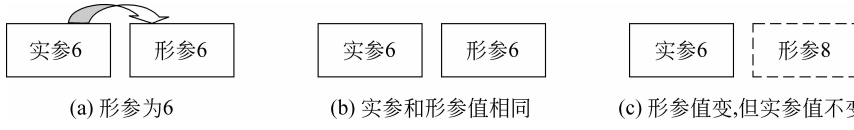


图 3-9　值传递的单向性

对于地址传送来说，由于虚参和实参实际上是同一个内存单元，所以可以认为数据的传送是双方向的。此时，虚参数值的任何改变都会影响实参的数值。如图 3-10 所示，过程调用时，实参的数值为 6，则形参的数值也为 6，若过程运行时形参的数值改变为 8，则过程结

束后主程序中实参的数值也就是8,而不是原来的6。

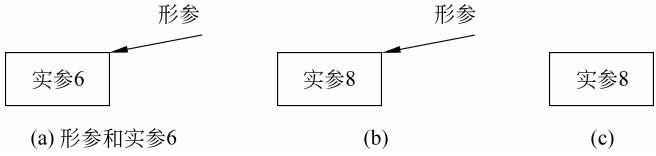


图 3-10　地址传递的双向性

参数的地址传送方式可以把过程的计算结果带回主程序中。前面说过,高级语言处理过程返回的第一种方法,是让定义为全局变量的过程名把过程的计算结果带回主程序中。有些复杂问题的计算结果有若干个,而过程名却只能带回一个计算结果。由于参数地址传送方式的双向性,可以把需要带回主程序的计算结果设计成地址传送方式的虚参,这样,在过程运行完后,主程序就可以从相应的实参得到过程的计算结果。

5. 过程调用示例

这里给出一个求1～n的累加和问题的完整C语言程序,来说明高级语言过程调用的实现方法。C语言程序如下:

```
int Sum1(int n)                                    //过程定义
{
    int i, sum;
    sum=0;
    for(i=1; i<=n; i=i+1)
    sum=sum+i ;
    return sum;
}

#include <iostream.h>
void main(void)
{                                                  //主程序
    int n, sum;
    printf("请输入数值 n:");
    scanf("%d"; &n);                               //输入 n
    sum=Sum1(n);                                   //过程调用
    printf("前 n 个数的累加和为 %d",sum);          //输出 sum
}
```

过程定义部分既定义了过程Sum1的具体实现,也定义了过程名Sum1是一个全局变量。主程序是以名字为main开始的部分,主程序中首先要求用户输入实参n的具体数值(假设用户输入100),然后以实参n=100调用过程Sum1,Sum1完成1～n的累加计算后,语句return sum实现把局部变量sum中保存的计算结果(数值为5050)传递给过程名Sum1的功能,然后返回主程序继续运行。因为过程名Sum1是一个全局变量,所以主程序中可以读取该变量中的数值。主程序中把全局变量Sum1中保存的计算结果(数值为5050)再赋值给主程序中的局部变量sum,最后主程序输出局部变量sum中的数值后结束程序运行。

3.2.7　注释语句

虽然程序是由计算机执行的，但程序是由人设计的，另外，当程序出现问题或需要改进时，需要人在理解原设计思想和设计方法的基础上做出新的改进。这样，对人来说，程序是否容易理解就显得非常重要。把容易被人理解的程序称做可读性强的程序。设计可读性强的程序的一个重要的基本方法是在程序中添加足够的注释信息，来说明设计该程序的思想和方法。因此，程序的基本元素包含注释语句。

不同的高级语言表示注释语句的方法不同。C 语言注释语句有以下两种表示方法：①//注释内容；②/ * 注释内容 * /。

例如，可以在 for 循环语句后面加一个注释内容为“循环语句”的注释语句，如下所示：

```
for(n=1; n<=100; n=n+1)sum=sum+n;          //循环语句
```

需要说明的是，程序中的注释语句只是为了提高可读性，计算机在执行程序时并不考虑注释语句。

3.3　常用高级语言

本节介绍目前最常用的高级程序设计语言的产生及特点。

3.3.1　C 语言

1. C 语言简介

C 语言最初的设计和实现是由 Dennis Ritchie 在 DEC PDP-11 上的 UNIX 操作系统环境下完成的，C 语言的前身是 BCPL 语言。很多年以来 C 语言的实际标准一直由 UNIX 操作系统第 5 版所支持。它的描述载在 Brian Keinighan 和 Dennis Ritchie 于 1988 年出版的《C 程序设计语言》这本书中。随着微型机的大众化，各种 C 语言的实用版本纷纷出现。在不同版本上，C 语言的源程序可以奇迹般地保持高度兼容。另一方面，由于没有一个标准，它们彼此之间也有一些差异。为了改变这种情况，美国国家标准局(ANSI)成立了一个委员会，他们在 1983 年夏初建立了一个 C 语言标准，并已在 1987 年被采用。目前，主要 C 编译器已经实现了 ANSI 标准。

C 语言允许对字节、位和地址直接操作，而地址是计算机的基本工作单位。C 语言也很便于移植，移植是指一个软件从一种计算机类型转移到另一种计算机类型仍能保持其功能。例如，如果一个在 Apple 上写的程序可以在 IBM PC 上运行，就可以说这个程序是可移植的。

所有的高级程序设计语言都支持数据类型这个概念。数据类型定义了一组值，可以通过一组操作来存取具有这种类型的变量。普通的数据类型包括整型、字符型和实型。尽管 C 语言有 5 种基本类型，但它并不像 PASCAL 或 Ada 语言那样对类型要求那么严格，C 语

言几乎允许所有类型互相转换。例如,在很多表达式中,C语言都允许整型和字符型自由混用,C语言也不进行像数组越界或者变量类型不匹配这一类的运行错误检查,这些类型的检查是程序员的任务。

C语言的特殊性表现在它允许对位、字节、字和指针的直接操作,这就使它很适合频繁使用这些操作的系统级程序设计。C语言的另一个特征是仅有32个关键字(Kemighan和Rimhie定义了27个,ANSI标准化委员会又增加了5个)。这些关键字构成了C语言的骨架与此对应的是,IBM PC上的BASIC语言有159个关键字。

2. C语言是一种结构化程序语言

C语言是块状结构的语言,并不是一种严格适用于C语言的说法,人们一般把C语言称为结构化程序语言,这是因为它们的结构很像ALGOL、PASCAL严格的块结构定义是指允许在过程或函数内部说明另外的过程或函数。在这种情况下,全局变量和局部变量的概念被作用域规则所代替,该规则控制变量或过程的可见性。因此,既然C语言不允许在函数内部建立函数,它也就不能叫做块状结构的语言。

结构化语言的一个显著性质就是代码和数据的分离。这种语言可以把为完成特定任务所用的信息和指令与程序的其他部分分隔开。所用的方法就是使用局部变量和子程序,利用局部变量,程序员可以写出对程序其他部分没有副作用的子程序。这样,就可以很容易地写出需要共享的代码段。

3. C语言是面问程序员的语言

有些语言如COBOL和BASIC就是面向非程序员的经典例子。COBOL语言的设计目标不是为了使程序员更自由,而是为了改善代码的可读性。它不关心代码的可靠性也不关心代码的执行速度,只是想让非程序员能读懂这个程序。BASIC语言最初也是为非程序员编写简单的计算机程序而设计的。与此相反,C语言完全是为程序员建立和使用的语言。C语言提供了程序员想要的一切:很小的限制,没有什么修饰,块结构,独立函数和很少的关键字。使用C语言可以使程序员达到使用汇编语言时的执行效率,同时还保留ALGOL和Modula-2的结构化特性。毫无疑问,在非常优秀的程序员中,C语言是一种很通用的程序设计语言。

C语言之所以在程序员之间能够流行的一个重要因素是它可以代替汇编语言。汇编语言用符号来表示计算机能够直接执行的二进制代码,每一条汇编语言的语句都对应着计算机实际执行的一个动作。尽管汇编语言可以给程序员完成其任务极大的灵活性和极高的执行效率,但却很难用汇编语言来开发和调试程序。并且由于汇编语言是非结构化的,最后写出的程序像一团乱麻,充满了转移语句、调用语句和变址语句,这种缺乏结构性的汇编语言程序很难阅读、改进和维护。更重要的是,汇编语言程序在不同的机器(CPU)之间不可移植。

4. C与C++语言

C语言是介于汇编语言和高级语言之间的语言,属于高级语言,是集汇编语言和高级语言的优点于一身的程序设计语言。1972年,C语言在美国贝尔实验室里问世。到了20世纪80年代,贝尔实验室在C语言的基础上推出了C++程序设计语言,成为最广泛使用的面向对象程序设计语言的代表。它适合作为系统描述语言,既可用来编写系统软件,也可用来编写应用软件。

C/C++ 具有很大的灵活性，但这是以开发效率为代价的。一般来说，相同的功能，C/C++ 开发周期要比其他语言长。人们一直在寻找一种可以在功能和开发效率之间达到更好平衡的语言。好的替代语言应该能对现存和潜在平台上的开发提供更高的效率，可以方便地与现存应用结合，并且在必要时可以使用底层代码。针对这种需求，微软推出了一种称为 C# 的开发语言。C# 在更高层次上重新实现了 C/C++，是一种先进的、面向对象的语言，通过 C# 可以让开发人员快速建立基于微软网络平台的应用，并且提供大量的开发工具和服务帮助开发人员开发基于计算和通信的各种应用。

3.3.2　C++ 语言

C++ 是既适合于作为系统描述语言，也适合于编写应用软件的既面向对象又面向过程的一种混合型程序设计语言，它是在 C 语言的基础之上发展起来的。

20 世纪 80 年代，美国 AT&T 贝尔实验室的 Bjarne Stroustrup 在 C 语言的基础上推出了 C++ 程序设计语言。由于 C++ 提出了把数据和在数据之上的操作封装在一起的类、对象和方法的机制，并通过派生、继承、重载和多态性等特征，实现了人们期待已久的软件复用和自动生成。这使得软件，特别是大型复杂软件的构造和维护变得更加有效和容易，并使软件开发能更自然地反映事物的本质，从而大大提高了软件的开发效率和质量。

C++ 越来越受到重视并得到广泛的应用，许多软件公司都为 C++ 设计了编译系统。如 AT&T、Apple、Sun、Borland 和 Microsoft 等，其中国内最为流行的应当是 Borland 公司的 Borland C++ 和 Microsoft 公司的 Visual C++ 。与此同时，许多大学和公司也在为 C++ 编写各种不同的类库，其中 Borland 公司的 OWL(Object Window Library)和 Microsoft 公司的 MFC(Microsoft Foundation Class)就是比较优秀的代表，尤其是 Microsoft 的 MFC，在国内外得到了较为广泛的应用。

3.3.3　Java 语言

Java 语言于 1995 年由 Sun 公司推出，是面向对象的程序设计语言。其语法与 C++ 很类似，但却简单得多。它充分吸取了 C++ 语言的优点，采用了程序员所熟悉的 C 和 C++ 语言的许多语法，同时又去掉了 C 语言中指针、内存申请和释放等影响程序健壮性的部分，使程序更加可靠、易懂。Java 语言具有安全、跨平台、面向对象、简单、适用于网络等显著特点，Java 语言已经成为流行的网络程序设计语言。Java 采用虚拟机技术，其源程序经编译生成字节代码后，需解释执行，并可在任何环境下运行，成为目前 Internet 上重要的程序设计语言。

1. Java 语言的历史背景

Java(爪哇)一词，取自南太平洋爪哇岛上一种咖啡的名字。提及咖啡，总是不禁使人想到优雅宜人的环境设计者起这样一个名字，也许想在如今这充满紧张和竞争的时代营造出一种轻松愉快的气氛与其他语言一样，Java 语言也是在一种偶然的情况下应运而生的。人们在异构网分布式环境下进行应用开发时面临以下三个方面的问题：

(1) 使用小部分系统资源进行应用传输的安全性；