Chapter 3

第 3 章　程序设计语言

3.1　程序设计语言的发展

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。对计算机工作人员而言，程序设计语言是除计算机本身之外所有工具中最重要的工具，是其他所有工具的基础。没有程序设计语言的支持，计算机无异于一堆废料。由于程序设计语言的这种重要性，从计算机问世至今的半个世纪中，人们一直在为研制更新更好的程序设计语言而努力着，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种，但这其中只有极少数得到了人们的广泛认可。

3.1.1　程序设计语言的划分

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 40 余年的历史，早已形成了规模庞大的系列。进入 20 世纪 80 年代以后，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。

程序设计语言发展到今天经历了机器语言时代、汇编语言时代和高级语言时代。从机器语言到汇编语言的发展，由于助记符的采用，增强了编程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，则增强了编程语言的自然性和提高了语言表达的效率。

1. 机器语言

计算机刚发明时使用的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。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，格式如下：

操作码	操作数

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在内存中的地址。

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
```

由此可以发现,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)都要人来完成。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。使用机器语言是十分痛苦的,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,必须另编程序,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,因此运算效率是所有语言中最高的。

2. 汇编语言

为了减轻使用机器语言编程的痛苦,人们进行了一种有益的改进: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,如用 ADD 代表加法,MOV 代表数据传递等。这样一来,人们很容易读懂并理解程序在干什么,纠错及维护都变得方便了,这种程序设计语言就称为汇编语言,即第二代计算机语言。

例如,上述计算 A=15+10 的汇编程序如下。

```
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,它用助记符号来表示机器指令的操作符与操作数(亦称运算符与运算对象)。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,专门负责将这些符号翻译成二进制数的机器语言,这种翻译程序被称为汇编程序。汇编程序就是完成这种转换工作的一种专门的程序。汇编程序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。汇编语言的问世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,编程工作变得容易多了。

机器语言中用术语机器指令来表示机器语言中某个特殊的操作。类似地,汇编语言中用术语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,用它编写哪怕是一个很简单的程序,也要使用数百条指令。为了解决这个问题,人们又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一些。为了解决由多人编写的大程序的拼装问题,人们又研制出了连接程

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。

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，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，程序精练而质量高，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. 高级程序设计语言

虽然用汇编语言编写程序较用机器语言编写程序方便了许多，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痛苦经历中，人们意识到，应该设计一种接近于数学语言或人的自然语言，同时又不依赖于计算机硬件，编出的程序能在所有机器上通用的语言。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。和汇编语言相比，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），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FORTRAN0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表达式的形式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助记符有本质区别的变量的概念，它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、语句和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是从低级语言的土壤中破土而出的，其许多成分中“低级”的痕迹随处可见。目前常用的FORTRAN版本是FORTRAN77与FORTRAN90。

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，曾经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和Java等。随着程序设计语言的不断更新和发展，许多高级语言由于先天的不足，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。

高级语言的出现大大推动了软件的发展，也是目前计算机得到广泛应用的一个重要原因。

3.1.2 高级程序设计语言的发展

高级语言的发展也经历了从早期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、ALGOL 和 COBOL。

2. 高级语言发展初期

FORTRAN、ALGOL60 与 COBOL 三种语言问世后，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，导致了程序设计语言数目呈指数般的激增。在 20 世纪 60 年代的 10 年中，人们至少研制了 200 多个高级语言。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 和 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和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。

(1) 函数式语言

函数式语言也叫作用式语言。纯函数式语言中不使用赋值语句，其语法形式类似于数学上的函数。典型的函数式语言有 LISP、APL 和 ML 等。

(2) 逻辑式语言

逻辑式语言也叫说明式语言，是基于规则式的语言，以逻辑程序设计思想为理论基础。其主要核心是事实、规则与推理机制，事实与规则用于表示信息(知识)，而推理机制则用于根据事实与规则产生执行结果，程序中不需要显式定义控制结构。其中最有代表性的逻辑式语言是 PROLOG。PROLOG 是由法国马塞大学人工智能研究室的 Roussel、Colmerauer 与英国爱丁堡大学人工智能系的 Kowalsik 合作于 20 世纪 70 年代初期研制出来的。自 1981 年日本政府宣布以 PROLOG 为基础的第五代计算机系统(FGCS)项目以来，PROLOG 已成为人工智能研究领域的主导语言。

(3) 面向对象式语言

面向对象式语言简称对象式语言,它与传统过程性语言的主要区别在于:在传统过程语言中把数据及处理它们的子程序当作互不相关的成分分别处理,而在对象式语言中则把这两者统一作为对象封装在一起进行处理。面向对象的思想是 G. Booch 在研究 Ada 软件开发方法时首先提出来的,而对象式语言中的一个重要概念是类,类最早是在 SIMULA67 语言中提出与实现的。但 SIMULA67 本身并不是一个面向对象的语言。完全面向对象的语言有 SMALLTALK、C++ 和 EIFFEL 等。

SMALLTALK 是 A. Kay 于 20 世纪 80 年代初研制出来的,他在研制这个语言之前已先期研制了 FLEX 语言。在 SMALLTALK 中对象之间通过消息进行通信。C++ 是在 C 的基础上扩充而成的,是 C 的超集。它在 C 的基础上扩充了类、对象、多继承和运算符重载等面向对象的概念。EIFFEL 是一个以可再用性与可扩充性作为其主要目标的对象式语言,于 20 世纪 80 年代中期问世。EIFFEL 是一个强类型语言,其主要成分是用于表示抽象数据类型的类。面向对象的概念与语言目前尚不完全成熟,还处于进一步发展中。

随着各种范型的程序设计方法的研究,各种范型的相互渗透,目前许多语言都不止体现了一种范型。例如,Modula. 2、Ada 等强制式语言中都体现了面向对象式范型的思想;C++、EIFFEL 等对象式语言中则充满了强制式语言的思想;LOGLISP 与 FUNLOG 兼有函数式与逻辑式范型的思想,LEAF 与 TABLOG 兼有关系式与逻辑式范型的思想,LEAF 与 TABLOG 兼有关系式与逻辑式范型的思想,LOOPS 语言则把函数式、对象式、逻辑式与存取式多种范型有机地结合在一起。在 20 世纪 80 年代中期问世的还有一个有名的多范型语言 Nial,它支持强制式与函数式范型,而新扩充的 Nial 版本还可以支持逻辑式与对象式范型。由于每一种范型尽管有其长处但也有其局限性,现在程序设计语言的一大发展趋势就是把各种范型的思想都尽可能地融合在一个语言中,以满足不同要求、不同风格的程序设计的要求。

3.1.3 高级程序设计语言的分类

高级语言有许多种分类方法。常用的高级语言分类方法有按照设计要求划分、按照应用范围划分和按照描述问题的方式划分等。其中,按照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. 程序设计语言按照用户的要求有过程式语言和非过程式语言之分

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 和 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、GPSS 语言等。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。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. 按照描述问题的方式分，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，其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、COBOL、ALGOL、BASIC、C、PASCAL、Ada 和 APL 等语言都属于命令型语言。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、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，计算机根据对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、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. 按照应用范围，有通用语言和专用语言之分

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 和 C 等。

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、ALGOL60、PASCAL 和 C 等。

5. 按照成分性质，有顺序语言、并发语言等之分

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 高级程序设计语言的基本构成

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、语法和语义等多种约定，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，假设有如下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，可以得出如下结论。

(1) 数据类型是高级语言为变量分配具体大小内存单元的需要。

(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、性别和年龄等，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。

由于数组变量是由若干个数组元素组成的整体,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、年龄和平均成绩等信息。我们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 3-1 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。

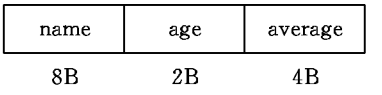

图 3-1 结构体变量 s 的内存单元分配

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。为与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。“=”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，因此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，因此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