●第一章　引　　论

程序设计语言与其它任何语言一样，也是由语法和语义两大基本成分构成的。为了精确地定义程序设计语言并有效地实现它，应用它，就必须对程序设计语言的各构成要素作深入的研究，以便找到严格定义语法和语义的好方法（也就是设计程序设计语言的好方法）。这本小册子的第二、三两章就讨论这方面的内容。作为这些讨论的准备工作，应当对语言、语法、语义、语用这些基本概念，对程序设计语言的概念，对判断什么是好的程序设计语言的准则，有一个基本的了解。引论的目的就是要给出有关这些概念的知识。

第一节　语言、语法与语义

当今世界里，电子计算机已经渗透到各个领域。成了现代科学技术、现代化的大工业生产，乃至现代化的生活所不可缺少的助手。现在，人们已经不再把电子计算机单纯地当作一种计算工具，而是把它看成一种几乎无所不能的信息处理工具，是可以在许多场合下代替人脑工作的电脑。计算机

究竟能否达到人的智能高度，这是颇多争议且难获解答的问题，但从信息处理的角度来看，计算机的出现却是实质性的一大进步，由此可以发展出许多新的观点。人们要利用计算机处理信息，首先就得以一定的方式表示信息，并把这些信息传送给计算机。一般地讲，这些信息表现为某种形式的语言。事实上，一个计算机系统就是一个能解释语言的机器。计算机接受语言的句子，理解它，服从它，根据它的要求完成一些动作，最后给出结果回答问题。可以毫不夸张地说，计算机系统完成其任务的整个过程都是建筑在以某种计算机语言所表达的程序（我们把为计算机规定的执行动作序列称为程序)的基础之上。这种把计算机看作是能解释语言的机器的观点决定了计算机语言在计算机世界里的重要地位，使计算机语言，特别是程序设计语言，成为计算机科学中人们兴趣最浓、研究最多、结果最富的领域之一。

作为一种语言，尽管是人造的语言，程序设计语言也服从语言学的一般规律，实际上程序设计语言与自然语言（英语、汉语等）也确有许多共性。由于这种缘故，在讨论程序设计语言之前，有必要弄清楚语言等几个更一般的概念。

语言

什么是语言？按照语言学上具有极大权威的 Webster 的定义,语言是‘为相当大的团体的人所懂得并使用的字以及组合这些字的方法的统一体”。这个定义基本上抓住了问题的要害：语言的意义（人所懂得的字）和形式（组合这些字的方法）的统一体，也就是语法和语义的统一体。

Webster 对语言的定义适宜于任何种类的语言（自然语言和人为语言），当然也适合程序设计语言（它只是人为语

言的一种）。但是，程序设计语言比自然语言简单得多，形式化得多，有关它的知识必须更精确（或者说更刻板，否则就不能为死机器所理解），所以，对于研究程序设计语言来说，这个定义显得太笼统。

为了更科学更严谨地研究语言的理论，人们发展了符号语言学（Semiotics，或译形式语言学），试图为语言（包括自然语言）的研究建立更形式的基础。在符号语言学中，为语言的概念作了远较形式而严格的规定。

为给语言下一精确定义，首先要有字母表或符号集的概念，这就是：符号的任何有穷集。虽然有不可数的无穷多个符号存在，但我们只考虑可数无穷子集，并从这样的子集中抽出我们要求的有穷集来。字母表可以包括数字，大写和小写的拉丁和希腊字母，$、£、＃等特殊等号，以及使用者希望有的任何符号。例如，

{0，1}

{A，B，…，Z}

{α，β，…，ω}

{人，饭，吃}

等都是合法的字母表。

然后,是句子的概念。一个字母表上的句子是由字母表的符号组成的有限长度的任何行。句子的同义词是行、串、字。句子称为语言的一个语法单位（有时又称为语法范畴），还可以有其他的语法单位，以此来表示符号构成的不同复杂程度的级别。例如，我们经常使用的分句、复句、短语、程序等都是这样的一些语法单位。任何字母表上均可以有的一个特殊句子是空句子，即不含任何符号的句子，记为 ε。设

V是一个字母表，我们用 V^* 表示由 V 的符号所组成的所有句子的集合，包括空句子 ε。另外，用 V^+ 表示集合 $V^* - \{\varepsilon\}$。因此，若 $V = \{0, 1\}$，则

$V^* = \{\varepsilon, 0, 1, 00, 01, 000, \cdots\}$

$V^+ = \{0, 1, 00, 01, 000, \cdots\}$ （ε不在V^+中）

在以上定义的基础上，我们可给语言下这样一个定义：

语言是一字母表上的句子的任何集合。

也就是说，对于字母表 V，V^*的任何一个子集均构成一个语言。比如说，上述全部二进制数的集合 $V^+ = \{0, 1, \cdots\}$ 就构成一个语言，我们称它为二进制数语言。

应当承认，就揭示问题的实质而言，这一定义并不比Webster的定义高明。第一，它全然没有接触到语言的语义。第二，有关语法，它也并没有告诉我们多少东西：句子的任何集合，这种限定确实也太空泛了些。不过话说回来，把这两个定义作这种比较也不太公平，因为语言的第二个定义本身就是对形式语言下的，自然是偏重语言的形式方面的抽象。而且，这一定义有两个明显的好处：

1. 这个定义的确是覆盖了任何语言。从最具人为色彩的二进制数的语言到最复杂的自然语言，例如汉语，都被网罗到这一概念之下。二进制数语言不用说了，汉语符合这一定义也容易看出，只要取字母表 V 为所有汉字（或许还要加上标点符号）的集合，则（书面）汉语显然是V^*的一个子集。不言而喻，本书的主要讨论对象程序设计语言也被这一语言定义所包容。

2. 这一定义足够地形式化，足够地精确，足够作为我们以后研究语言的基础，使我们能在这一严格的基础上发展

出一批知识。

在以上讨论的基础上，我们就可以明确确定本书对程序设计语言的研究所采取的立场了。这就是：以语言是语法和语义的统一体这一基本事实为原则，把程序设计语言看成字母表上句子的集合而将其纳入更一般的形式语言范畴之内，以便运用形式语言理论的结果处理程序设计语言的语法问题，进而在语法研究的基础上讨论其语义。

下面来谈语言的语法和语义。在阐述这两个概念时，摘引一下 Morris对语言三要素的定义是很有益的。

符号语言学的创始人，美国哲学家 Charle Morris 把对符号语言的研究分为语法、语义、语用三部分。在他的《符号、语言和行为》一书中，Morris 是这样定义语言的三个要素的：

语用——关于在某种行为中发生的语言符号或信号的由来、使用和效果。

语义——关于语言符号或信号在其所有表现方式中的意义。

语法——有关语言符号或信号的组合方式而不涉及这些符号或信号所表达的意义，也不涉及这些符号或信号与它们发生于其中的行为之间的关系。

这种明确地把语言划分为语法、语义、语用三个要素，通过对它们的分别研究来把握语言整体的方法已被大多数语言学者奉为圭臬。应当说，Morris的语言三要素抽象也确实是客观地反映了现实中各种语言的属性。以下我们就来分别讨论语言的这几个要素。

语法

与汉语的“语法”这个词相对应的英语单词有两个：“*syntax*”和“*grammar*”。一般情况是，对于自然语言的语法，我们使用 grammar，而在谈论程序设计语言这类人为语言时就使用 syntax。不过应当指出的是，对于计算机语言学者，*syntax*和 *grammar* 还有另一层细致的区别。grammar 通常用来指控制语言中行的产生的规则，而 *syntax* 通常被看成计算机用以辩识一给定行是否为语言中合法行（有时我们把这种合法行称为语言的一个实例）的规则。这个问题，以后我们还有机会作更深入的讨论。这里我们使用的语法（syntax）一词就是笼统地指语言（特别是程序设计语言）的形式。而把 grammar 译为“文法”，意指形式语言学中的形式文法。

语法，即语言的形式，它表现为字符以及由字符构成更大的语法单位（如句子）的规则。根据这些规则就可以辩识出语言的哪些构成形式是语法上可接受的。换句话说，语法决定了一语言的一切合法构成。

设如我们有这样的一种“汉语”，其中有这样的两条语法规则：

“一个

名词　动词　名词

的序列就是一个句子”。　　（1 a）

“字符

人

饭

是名词，

吃

是动词”。 (1 b)

那么，

人 吃 饭

饭 吃 人

均为这种汉语的合法句子。尽管我们知道，只有第一句才于理相合而第二句子是有背情理的，但这里我们讲的是语法，只讲形式而不管意义。

第二个例子是前面已经提到过的二进制数语言。可以简单地给这个语言作如下的语法定义：

“字符 0 和 1 的一个有限序列构成句子。”于是

0

1

101

10101

等都是二进制数语言中的合法句子。至于 101 和 10101 究竟是什么东西，我们暂时全不知晓。

总而言之，应当始终记住的是，决定一语言结构是否能合法地接受的唯一判断标准是语法规则。

关于语法上正确的某种语言构成形式（比方说，句子），其意义可以从两种角度来看。这就是，说（或写）句子的人意图表达的意义与听（或读）句子的人所理解的意义。前者就是语言的语义（semantics），后者就是语言的语用（pragmatics）。大多数情况下这两种意义总是相同的，否则的话，人们就不能利用语言来有效地交流思想了。但这两种意义又并非总是相同的，于是乎才可能有误解发生。在幽默作品和相声演出中，可以找到故意使语义和语用意义不一致从而闹

出笑话的大量例子。由此可知，把语义和语用区分开来对待是恰当的。下面让我们先来看看语义。

语义

简单地说，语义就是与语言的形式相联系的意义。既然如此，在规定某语言成分的意义时也就绝对离不开该成分的构成规则。换言之，规定语义时必须以语法作为基础。例如对于上述“汉语”一例中的句子

人　吃　饭

必须要在语法规则(1 a)和(1 b)的基础上来规定其语义。

根据(1 b)规定名词代表客体，动词代表动作。

根据(1 a）规定前一名词代表的客体把动词代表的动作作用于后一名词代表的客体。

这样一来，句子

人　吃　饭

的意义也就确定了。

关于二进制数语言的例子，情况也一样。对这种数的通常解释是：每个二进制数表示一个自然数。例如

101　　表示自然数 5

10101　表示自然数21

通过这种使语言的每个成分对应于一个熟知的数学对象的方法来定义语言的语义。在这里，如何具体对每个二进制数，确定出它所表示的自然数，也就是说如何对整个语言，而不是对语言的某特定结构给出其语义定义，那又必须借助于语法规则了。在以后的讨论进程中，我们还会回到这个问题。

语用

关于语用，前面已经说过，它可以简单地看成语言的接

收者所理解的意义。由此可以推想，对于自然语言和程序设计语言这样的人为语言，语用的影响无疑会有所不同。今针对我们的主要研究对象程序设计语言来讨论一下这个问题。一方面，程序设计语言具有人与人之间交流算法等方面的目的，因而有可能发生写程序的人意欲表达的意义被读程序的人误解这种情况。语义和语用的这种不一致可以通过语言的标准化、精确的语法和语义定义,以及正确的程序设计逻辑和良好的程序设计风格（主要指使用记忆标识符、模块化、完整的内部程序文件等用以提高程序可读性的技术）来排除。另一方面，也是更主要的一方面，程序设计语言又是人与计算机通讯的工具，是要被计算机所理解的语言，那么，这种情况下的语用影响又表现在什么地方呢？计算机是死的机器，它对语言的理解方式实际上是人强加给它的，因此我们总可以使计算机对语言的理解与语言本身表达的意义相吻合。换句话说，对于作为人机通讯工具的程序设计语言，我们总是可以作出安排，使语义和语用没有什么差别。但随之而来的是，要怎样才能办到这一点呢？显而易见，这已经完全是如何实现语言的问题了。从以上讨论可知，正如Morris对语用下的定义那样，“关于在某种行为中发生的语言符号或信号的由来，使用和效果”，程序设计语言的语用牵涉到语言的实现技术、程序设计方法学，以及语言开发和使用的历史诸多方面。由于对构成语用的这些问题的讨论是计算机科学中专门的研究领域，是需要大部头专著论述的课题，因此本书基本上不涉及语用学。只是在讨论语言的语义时，于必要之处注意一下语用的影响。比如，我们在讨论二进制数语言时，无论语法和语义定义得如何严格，但要实现

这个语言，把二进制数表示于定长存贮器之中时，我们就不得不考虑到语用的影响了。因为，由于存贮器容量的限制，是绝不可能对此语言中的任意数照语法和语义所规定的那样实现的。

在结束本节之前，我们打算略略提一下语言的二义性问题。对于这个程序设计语言的重要问题，后文将更详细地讨论。

二义性

所谓语言的二义性，顾名思义，就是可以把同一个语言表示解释成二个以上的含义。语言的二义性可以由语法的不适当定义引起，可以由语义的不适当定义引起，也可以由语义和语用意义间的混淆引起。

自然语言中充满着数不清的二义性语言表示。特别是，在口语中，语义和语用意义常有差别。重读、语气的变化等都会导致二义性。在广告标牌等简缩使用语言的场合，也常常可以见到有二义性的语句。例如，在飞机上就钉有写着如下英语语句的牌子：

NO SMOKING AREA IN REAR CABIN

这句话究竟意味着

后舱中有一不允许吸烟的区域，　　(1 c)

还是指

在后舱中没有允许吸烟的地方　　(1 d)

呢？这一语句的二义性显然是由于没有正确地使用连字符引起的。如果要把该语句解释成(1 c)则应写为，NO-SMOKING，如果要解释为 (1 d)，则应写为 NO SMOKIND-AREA。这是语法二义性的一个例子。

对于程序设计语言，由于上文讨论语用时指出的那些原因，我们不关心语义和语用意义的混淆引起的二义性。比起语义二义性、语法二义性更为重要，研究所得结果也较多，是我们的主要注意所在。

第二节　程序设计语言

要对程序设计语言作进一步研究，首先应当弄清楚什么是程序设计语言。

与计算机科学中的许多概念一样，程序设计语言至今（虽然已经被人们使用了30多年）未能有一个公众一致同意的严格定义。以下是好些种定义中常见的两个。

1．用于编写计算机程序的语言。

2．一套描写算法和数据结构的记法。

定义1比较广泛，包括了机器语言等所有计算机语言。定义2是N·Wirth 的著名公式：

算法 +数据 =程序

的另一种表达形式，抓住了实质，但不够具体。

本书中，我们把程序设计语言和高级语言二者视为同一概念，并给出如下的定义。这个定义首先对程序设计语言的概念作较一般的规定，然后加上一些解释性的说明来进一步限定它。

程序设计语言

程序设计语言是一组字符，以及确定如何把这些字符组成具有某种可供计算机理解其意义的结构的规则。它区别于其它计算机语言（如机器语言、汇编语言）的特点有如下四

个

1．程序设计语言的用户无须具有机器代码的知识。这就是说，一旦用户学会了某种特定的程序设计语言，他就可以在适当的机器上使用这种语言而不必还要再去了解该机器的机器语言。当然，这样说并不意味用户可以完全不管实际的机器。比如用户也可能需要知道在他使用的机器中浮点数是如何表示的，也还须知道机器有哪些配置，以便能充分利用机器提供的各种资源来编写出更有效的程序。但这里的基本之点在于：不能要求高级语言的用户同时必须具有关于机器代码的知识基础。唯其如此，高级语言才有存在的价值。

2．程序设计语言必须要有一定的机器独立性。换句话讲，必须要有充分的潜在可能性，使能在一台机器上运行的高级语言程序毋须完全重写（可能要求一些修改）就能在另一机器上运行。这条性质使重复劳动的减少成为可能，也保证了程序设计语言应用的广泛性。

3．当把程序设计语言翻译成机器语言时，一般来说，对于前者的一个可执行单位（例如一条可执行语句）应有多于一条的机器指令与之对应。这一要求使高级语言的用户能更容易地编出紧凑简练的程序。进而言之，高级语言的面向过程、面向问题的特点也深深地扎根于这一属性（以及下面的性质 4）之中。

4．解决某类问题的程序设计语言，其所使用的表现形式一般来说应接近该类问题在本学科中的自然表述形式（例如，适于作科技问题的数值计算的 FORTRAN，其表达式与数学公式接近）。这就使高级语言的表现形式远比完成同样功能的机器指令序列更好理解，使高级语言更适宜于描述间

题，更易于读、写和交流。

知道了什么是程序设计语言以后，我们要进一步问：一个好的程序设计语言应当符合什么样的要求？如何才能设计出好的程序设计语言？为回答这两个问题，我们先将程序设计语言与自然语言作一比较。

自然语言和人工语言都是信息交换的工具，但它们有一个明显的区别。自然语言的语法（及语义）是从遥远的古代起，在人们的长期实践中自然而然地形成的，是约定俗成的，它是历史的沉积物，是群众智慧的结晶。任何一种自然语言，它的存在，就是它能起到信息交换工具的作用的明证。因此，不管自然语言里面有多少似乎是“不合理”的语言规则，有多少含混不清的用法，人们也只好容忍它。最多不过针对某些问题呼吁两声，提几个改革方案，希望使语言的使用更加‘合理”。但是,如不经过时间的审查,任何试图对语言规则的些许修改均不能成立。而人工语言的语法和语义却是人为地规定的。人工语言有固定的使用范围和确定的使用目的，它的定义应以其使用范围和目的为依据。程序设计语言有双重的使用职能。一方面，它是人与计算机间交换信息的工具：程序一般是为了被计算机所理解与执行而编写的；另一方面，它又是人与人之间交换信息的工具：程序员用它来交流算法。因此，程序设计语言的定义既要面向人又要面向机器。程序设计语言设计得好不好就要看它是否能够很好地完成这个双重职能。根据这种观点，我们可以给好的程序设计语言提出如下的一些要求。

首先，为了使语言很好地起到人与人之间交换信息的作用，我们要对它提出以下两条面向人的要求：易读性和易写

性。

易读性

语言的一个实例，即程序，是易读的，是指由程序表示的算法和数据是清楚明白一目了然的。通常，具有很高易读性的程序应当是自成文的，就是说，即使不附加其它任何文档资料，也能理解程序的意思。不过，在实际情况中，这么高的易读性很难达到。在语言的定义中，自然语言的格式、结构语句、关键字和无含义字的随便使用、可以插入注释、不限定标识符的长度、助记忆的运算符符号、自由域的格式、完整的数据说明等这些语言特点都能提高易读性。而提高易读性的一个最有效的办法，则是在语言的设计中，对基本语义不同的语言结构，采用不同的语法规则。做类似事情的程序结构看上去是类似的；做完全不同事情的程序结构看上去是不同的。这一语言设计思想是提高可读性的好办法。例如，大多数语言中，不同的语句类型采用不同的语法结构，因而把条件转移、循环以及转向控制结构之间的区别弄得清清楚楚。通常，使用的语法越复杂，程序结构就越容易反映不同的基本语义结构，从而有助于程序的理解。提供少量不同语法结构的语言，通常也就只能写出不易读的程序。 例如APL中仅提供一种语句格式，对于赋值语句、子程序调用、无条件转向、子程序返回、多路条件转移以及其它普通的程序结构间的语法区别只能通过复杂表达式中一个或几个不同的运算符来反映，甚至判定程序的总体结构也常常不得不详细地分析整个程序。所以，用 APL很难写出易读的程序。由此可知,语言的设计是程序可读性的前提,如果语法的设计没有蕴含语言的高度可读性，则最好的程序员也用它写不出易读的

程序。但是，语言的设计并非是程序易读的保证，即使设计得具有极高可读性的语言,到了低能的程序员手中,也可能编出不易读的程序。不过,尽管程序的易读性不是单纯可以由语言的语法和语义定义决定的，我们仍然把易读性要求作为程序语言定义时应考虑的一条重要原则，因为它是编写易读的程序的前提，因而也就是人与人之间有效地交流信息的前提。

易写性

信息交换是涉及送方和受方双方面的过程。易读性是从信息的受方的角度提的要求，而易写性则是从信息的送方的角度提的要求。程序的易写的语法特点经常是和程序的易读的语法特点相矛盾的。使用简洁和有规则的语法结构可以提高易写性，而易读性的提高却要求使用各种更繁琐的结构。例如，容许说明和运算有不加指定的隐式语法规定使得程序较短 和容易书写，但读起来又较难了。当然，也有些语法特点既能帮助提高易写性，又能帮助提高易读性。例如前面提到的，结构语句的使用，简单的自然语句格式的使用，助记忆的运算符符号的使用，无限定的标识符的使用等等即是。很明显，语言的易写和易读是它能否争取到大量用户的一个重要因素。

易实现

前述两条要求是面向人的，是程序员直接需要的标准。而易实现则是面向计算机的，是从计算机理解语言的角度提出的要求,而这一要求也往往是与易读和易写相冲突的。易实现就是要求语言所写的程序便于翻译成可执行的形式（或容易解释执行），而便于翻译的关键在于语法结构的规律性，

因此象允许自然语言的格式书写、不限定标识符的长度、自由域格式书写等有助于提高语言易读性（或易写性）的语言特点都或多或少会增加翻译的难度。特别是，如为了提高易读性而把语法搞得很复杂，则势必使语言难于实现。例如COBOL语言，尽管它的语义不怎么复杂，但是却包含了大量的语句和说明的形式，实现起来就变得很困难。 LISP语言的语法提供了既不易读也不易写，却便于实现的程序结构的范例。之所以这样，就是因为 LISP 的语法结构能用少量的简单规则来描述，具有很强的规律性。由于我们一般总是要求程序设计语言是能在机器上实现并付诸使用的,因此,对语言的易实现的要求也是在定义语法时必须认真考虑的。

最后，我们要提出一个面向人和机器两者的要求。或许，这也是语言设计中最重要的一个要求，这就是无二义性。

无二义性

理想的语言定义对程序员所能写的每一种语法结构只提供唯一的意义，而二义性的结构却允许有两种或更多的不同的解释。一个具有二义性的语言，不管是作为人与人之间，亦或人与机器之间交换信息的工具，在使用的过程中显然都会引起一些问题。故程序语言的设计应尽量免避二义性。第一章中我们已说过，语法二义性是语言设计中应放在首位解决的问题，这里我们要对语法二义性稍加说明。

我们说一个语言有语法二义性，是指这个语言中存在这样的句子，这种句子可以分析成两种以上的不同的语法树。我们举一个例子来对之进行解释。

二义性问题一般不出现在单个的程序单位结构中，而是出现在不同结构间的相互关系之中。不少语言都允许两种不

同形式的条件语句，例如：

if B_1 then st_1 else st_2

if B_1 then st_1

在单独使用它们时都不会产生二义性，但若把两种形式的语句结合起来使用，使一种条件语句中的st_1是另一个条件语句时，就会产生具有语法二义性的如下结构：

if B_1 then if B_2 then st_1 else st_2

它可以分析为以下的两种语法树。

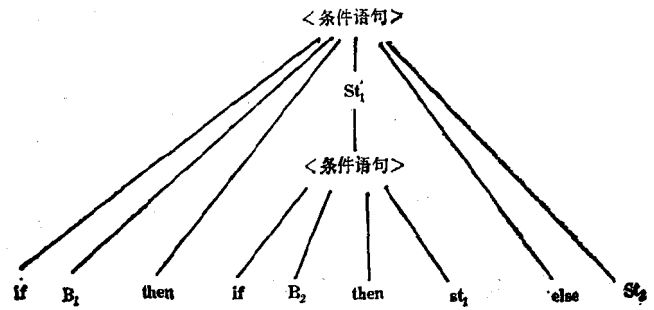
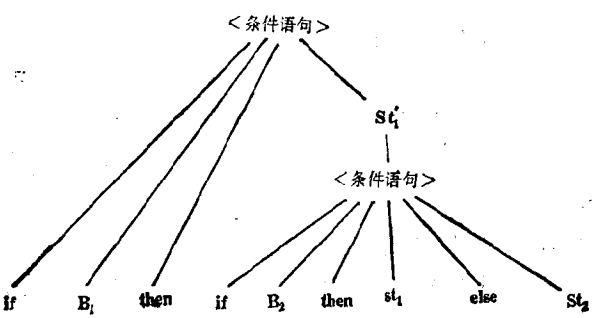


图1 (a)



(b)