# 编译原理复习材料

## 第一章绪论

1、计算机执行用高级语言编写的程序有哪些途径？他们之间的区别是什么？

在计算机上执行用高级语言编写的程序有两种途径：（1）编译，即用一个编译程序把高级语言翻译成机器语言程序；、，然后再运行所得的机器语言程序求得计算结果；（2）解释执行，以高级语言写的源程序作为输入，不产生目标程序，边解释边执行源程序本身。两种方法的主要区别在于，第（1）种方法会产生目标程序，但第（2）种方法不会。

2、什么叫编译程序？

通常所说的翻译程序是这样的一个程序，它能够把某一种语言程序（称为源语言程序）转换成另一种语言程序（称为目标语言程序），而后者与前者在逻辑上是等价的。如果源语言是诸如 FORTRAN、Pascal、C、Ada、Smalltalk 或 Java 这样的“高级语言”，而目标语言是诸如汇编语言或机器语言之类的“低级语言”，这样的一个翻译程序就称为编译程序。

3、编译程序的工作过程

编译程序的工作，即从输入源程序开始到输出目标程序为止的整个过程是非常复杂的。通常，编译程序的工作过程可以划分为 5 个阶段：

第 1 阶段，词法分析。词法分析的任务是输入源程序，对构成源程序的字符串进行扫描和分解，识别出一个个的单词（亦称单词符号或简称符号），如保留字（基本字）、标识符、算符、界符等。

**依循的原则：构词规则**

**描述工具：有限自动机**

第 2 阶段，语法分析。语法分析的任务是在词法分析的基础上，根据语言的语法规则，把单词符号串分解成各类语法单位（语法范畴），如语句、程序块、函数等，判断整个输入串是否是一个语法上正确的“程序”。

**依循的原则：语法规则**

**描述工具：上下文无关文法**

第 3 阶段，语义分析与中间代码产生。这一阶段的任务是对语法分析所识别出的各类语法范畴，分析其含义，并进行初步翻译（产生中间代码）。通常的中间代码有三元式、四元式、间接三元式、逆波兰式等。

**依循的原则：语义规则**

第 4 阶段，优化。优化的任务在于对前段产生的中间代码进行加工变换，以期在最后阶段能产生更为高效（省时间和空间）的目标代码。优化的主要方法有公共子表达式的提取、循环优化等。

**依循的原则：程序的等价变换规则**

第 5 阶段，目标代码生成。这一阶段的任务是把中间代码（或经优化处理之后的中间代码）变换成特定机器上的低级语言代码。

**依赖于硬件系统结构和机器指令的含义**

目标代码三种形式：

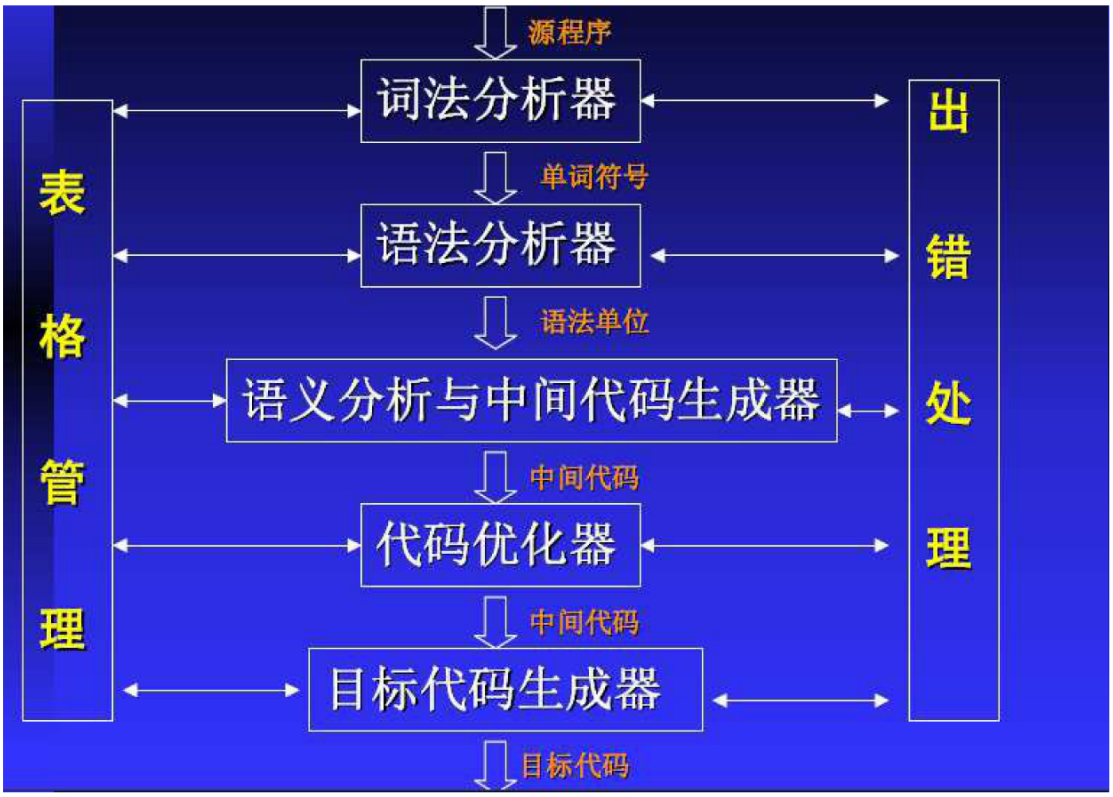
绝对指令代码：可直接运行

可重新定位指令代码：需要连接装配

汇编指令代码：需要进行汇编

4、编译程序在分析语言程序的过程中将源语言程序的各种信息都保留在各种表格中，同时，在源语言程序出错时，编译程序要帮助进行错误判断、错误定位等，因此表格管理及出错处理和编译程序的各个阶段都有关联。

编译程序的结构图如图所示。




编译程序结构

5、什么叫遍？遍和阶段有什么不同？

所谓"遍"， 就是对源程序或源程序的中间表示从头到尾扫描一次。

阶段与遍是不同的概念。一遍可以由若干段组成，一个阶段也可以分若干遍来完成。

遍与阶段的含义毫无关系。

6、什么叫自展？什么叫交叉编译？

采用“自编译方式”产生编译程序的方法叫自展。即先对语言的核心部分够造一个小小的编译程序（可用低级语言实现），再以它为工具构造一个能够编译更多语言成分的较大编译程序。如此扩展下去，最后形成整个编译程序。一般称运行编译程序的计算机为宿主机，运行编译程序所产生目标代码的计算机称为目标机。如果编译程序对编译时产生不同于其宿主机的机器代码，则称它为交叉编译，即在 A 机上的编译程序对高级语言编译后所产生的目标代码是 B 机的机器代码，而且 A 机和 B 机使用不同的机器代码。

# 第二章高级语言及其语法描述

## 1、程序语言的定义

一个程序语言是一个记号系统。如同自然语言一样，程序语言主要是由语法和语义两方面定义的。有时，语言定义也包含语用信息，语用主要是有关程序设计技术和语言成分的使用方法，它使语言的基本概念与语言的外界（如数学概念或计算机的对象和操作）联系起来。任何语言程序都可以看做是一定字符集（称为字母表）上的一个字符串。合乎语法的字符串才算是一个合法的程序。

语言的语法是用来形成一个合法程序的一组规则。这些规则的一部分称为词法规则，另一部分称为语法规则（或产生规则）。语言的词法规则是指单词符号的形成规则。语法规则则规定了如何从单词符号形成更大的结构（即语法单位），因此语法规则也可以说是语法单位的形成规则。

词法规则和语法规则定义了程序语言的形成规则，而程序的意义则用语言的语义规则来定义。语义规则规定了语言的单词符号和语法单位的意义，是定义一个程序意义的一组规则。

## 2、高级语言的分类

根据其功能和形成规则，高级语言可以分为一下几类：

（1）强制式语言，也称过程式语言。其特点是命令驱动，面向语句。一个强制式语言程序是由一系列的语句组成的，每个语句的执行引起若干存储单元中的值的改变。如 FORTRAN、C、Pascal，Ada 等等。

（2）应用式语言，也称函数式语言。这类语言更注重程序所表示的功能，而不是一个语句接一个语句地执行。程序的开发过程是从前面已有的函数出发构造出更复杂的函数，对初始数据集进行操作直至最终的函数可以用于从初始数据计算出最终的结果。如 LISP、ML。

（3）基于规则的语言，也称逻辑程序设计语言。这类语言的程序执行过程是：检查一定的条件，当它满足直，则执行适当的操作。如 Prolog 等。

（4）面性对象语言，是如今最流行，最重要的高级语言。它的主要特征是支持封装性、继承性和多态性等。把复杂点的数据和用于这些数据的操作封装在一起，构成类；通过类的继承和复合设计出更复杂的类。如 Smalltalk，C++，Java 、面向对象的 PASCAL 等。

## 3、数据类型和操作

程序的本质是描述一定数据的处理过程。因此对于大多数程序设计语言而言，数据的概念是最基本的。程序设计语言所提供的数据及其操作对语言的适应性有很大影响。一个数据类型通常包括以下 3 种要素：

（1）用于区别这种类型数据对象的属性；
（2）这种类型的数据对象可以具有的值；
（3）可以作用于这种类型的数据对象的操作。

一个程序语言必须提供一定的初等数据类型，包括这些数据类型上能进行的运算的定义。不同的语言含有不同的初等数据成分。常见的初等数据类型有：

（1）数值类型：整型、实型、复数、双精度，　　运算：+，-，*，/等
（2）逻辑类型：布尔运算：∨，∧，¬
（3）字符类型：符号处理

（4）指针类型：指针式把内存地址作为其值的数据类型，通过指针可以操作内存空间。

程序语言中的各种名字都是用标识符表示的。标识符是指由字母、下划线和数字组成的，以字母或下划线为开头的一个字符串。名字和标识符在形式上难于区别，标识符是一个没有意义的的字符序列，而名字则有明确的意义和属性。用计算机术语来说，每个名字可看成是代表一个抽象的存储单元，这个单元可含有一位、一字节或相继的多个字节。该单元的内容则被认为是名字的值。仅把名字看成代表一定的存储单元还是不够的，我们还必须同时指出它的属性（数据类型）。只有指定了属性的存储单元，其值才是可以理解的。

一个名字的属性包括类型和作用域。名字的类型决定了它能具有什么样的值， 值在计算机内部的表示方式，以及对它能施加什么运算。名字的作用域规定了它的值的存在范围。

除了初等数据类型外，有些语言还提供了由初等数据构造复杂数据的手段。常见的复杂数据类型有：

（1）数组。一个数组是由同一类型数据所组成的某种 n 维矩形结构。数组在内存中占有一块连续的空间，系统采用基地址加偏移量的方式来访问数组元素。

（2）记录。从逻辑上讲，记录是由已知的数据组合起来的一种结构。一个记录通常含有若干个分量，每个分量称为记录的一个栏（或域 field）。每个分量都是一个确定类型的数据，不同的分量的数据类型可以不同。

（3）字符串、表格、栈和队列。

（4）抽象数据类型。抽象数据类型封装了数据和操作，在面向对象程序设计语言中，Ada 通过程序包（package）提供了数据封装的支持，Smalltalk、C++和 Java 语言则通过类（class）对抽象数据类型提供支持。

4、函数调用的方式

程序是由函数或过程构成的，程序的任务是通过函数或过程之间的协作（相互调用）1 来完成的，函数或过程的调用有以下 4 种方式：传地址（call-by-reference）、得结果（call by result）、传值（call-by-value）和传名（call by name）。

定义函数 Swap，其中 M、N 是形式参数，简称形参。

```
Void Swap(int M，int N)
 {
   int t;
  t=N;
  N=M;
  M=t;
}
```

函数调用：

Swap（I,J）；其中 I 和 J 是实在参数，简称实参。下面我们以下面的程序为例分别讨论 4 种参数传递的方式。

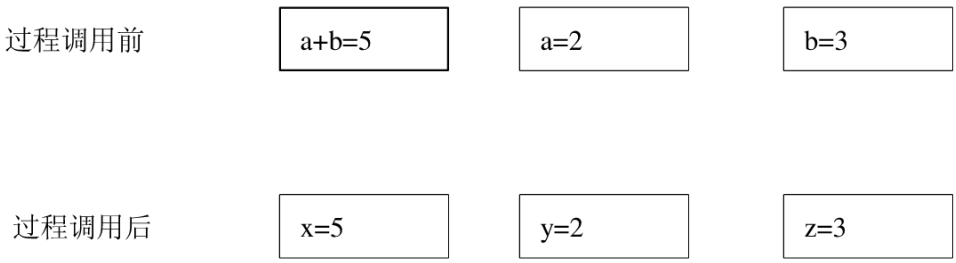
考虑下面的程序：

```
    …
procedure P(x, y, z);
    begin
      y:=y+1;
      z:=z+x
    end;
    begin
      a:=2;
      b:=3;
      P(a+b, a, a);
      print a
    end.
```

试问，若参数传递的方式分别采用传值、传地址、得结果和传名时，程序执行后输出 a 的值是什么？

（1）所谓传值是调用段把实在参数的值计算出来，被调用段把这些值抄进自己的形式参数中，像使用局部名一样使用这些形式单元。对形式参数的任何运算不影响实参的值。

上面过程 P 调用的的参数传递过程如下图所示。

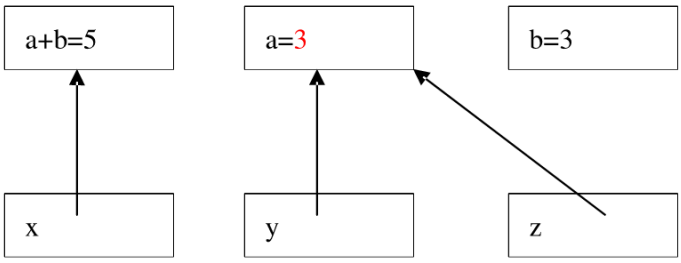



但过程 P 无法改变实参 a 的值。因此，程序执行后输出 a 的值是 2。

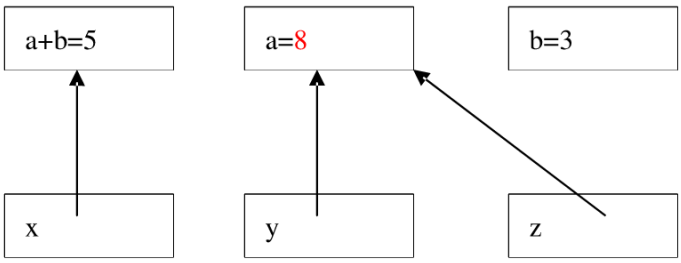
（2）所谓传地址是把实在参数的地址传递给相应的形式参数。在过程段中每个形式参数都有一个相应的单元，称为形式单元。形式单元将用来存放相应实在参数的地址。过程体对形式参数的任何引用或赋值都被处理成对形式单元的间接访问。

过程调用后，形参 x 的形式单元指向存 a+b 值的临时变量的地址，形参 y 的形式单元指向变量 a 的地址，形参 z 的形式单元指向变量 b 的地址。形参通过指针可以间接访问实参。

执行 y:=y+1; 后，实在参数的变化：

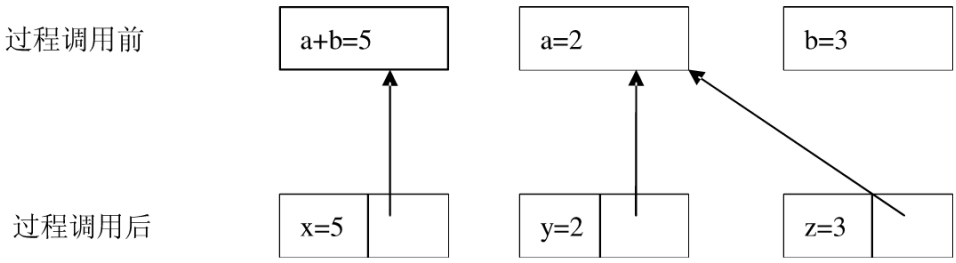



执行 z:=z+x; 后，实参的变化：

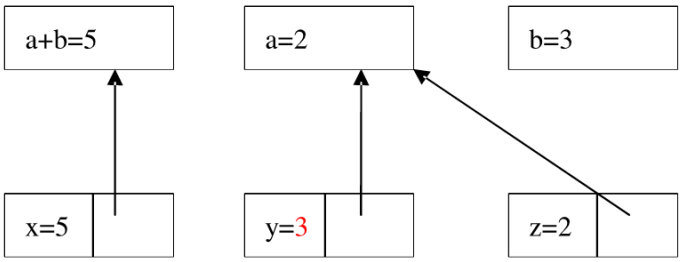



因此，程序执行后输出 a 的值是 8。

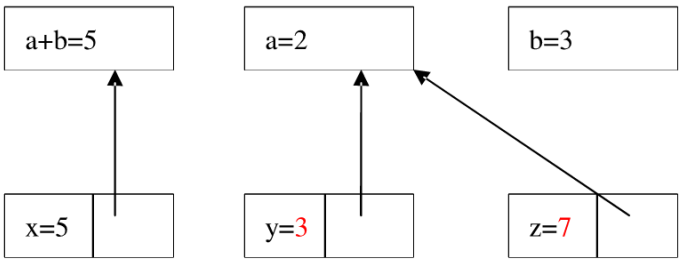
（3）所谓得结果是每个形式参数对应两个单元，第一个单元存放实参的地址，第二个单元存放实参的值。在过程体中对形参的任何引用或赋值都被处理成对第二个形式单元的直接访问，但在过程工作完成返回之前必须把第二个单元的内容存放到第一个单元所指的那个实参单元之中。



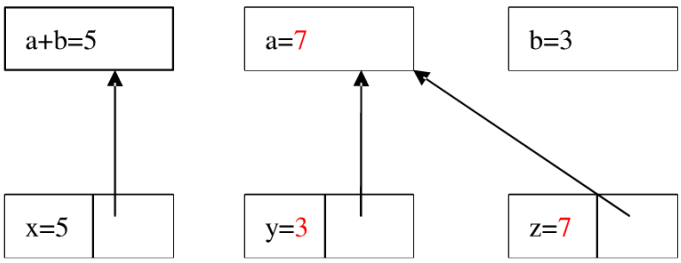

执行 y:=y+1; 后，实参和的形参的变化：




执行 z:=z+x; 后，实参和的形参的变化：




过程工作完成返回之前必须把第二个单元的内容存放到第一个单元所指的那个实参单元之中。实参和的形参的变化：




因此，程序执行后输出 a 的值是 7。

(4) 所谓传名是在进入调用段之前不对实在参数预先进行计值，而是过程中每当使用到相应的形参时才对它实行计值。因此,在实现时通常都把实参处理成型个子程(称为参数子程序),每当过程体中使用到相应形参时就调用这个子程序。

因此，过程体执行 y:=y+1;语句，实现时处理成为：

a=a+1;

过程体执行 z:=z+x;语句，实现时处理成为：

a=a+(a+b);

执行上述两语句后，a 的值是 9。因此，程序执行后输出 a 的值是 9。

综上所述程序执行时 a 的输出：

(1)传值：2　(2)传地址：8　(3)得结果：7　(4)传名：9

5、几个基本概念

考虑一个有穷字母表Σ字符集,其中每一个元素称为一个符号,Σ上的字(也叫字符串)　是指由Σ中的字符所构成的一个有穷序列,不包含任何字符的序列称为空字，记为 ε ,用Σ*表示Σ上的所有字的全体,包含空字 ε

例如: 设　Σ={a，　b}，则　Σ*={ ε ,a,b,aa,ab,ba,bb,aaa,...}

Σ*的子集 U 和 V 的连接（积）定义为

$$UV=\{\ \alpha\beta \mid \alpha\in U\ \&\beta\in V\ \}$$

例如设：$U=\{\ a,\ aa\ \}$　　$V=\ \{\ b,\ bb\ \}$

那么：

$UV=\{\ ab,\ abb,\ aab,\ aabb\}$

V 自身的 n 次连接（积）记为

$V^n=\underbrace{VV\ldots V}_{n\text{ 个}}$

规定 V0={ε}，令 V*=V0∪V1∪V2∪V3∪…

称 V*是 V 的闭包;记 $V^+=VV^*$ ，称 V+是 V 的正规闭包。

设：$U=\{\ a,\ aa\ \}$ 那么：

$$U^*=\{\ \varepsilon,\ a,\ aa,\ aaa,\ aaaa,\ \ldots\}$$

$$U^+=\{\ a,\ aa,\ aaa,\ aaaa,\ \ldots\}$$

闭包 V* 中的每个符号都是由 V 中的符号串经有限次连接而成的。

6、上下文无关文法

文法是描述语言的语法结构的形式规则即语法规则。这些规则必须是准确的，易于理解的，而且，应当有相当强的描述能力，足以描述各种不同的结构。所谓上下文无关文法是这样的一种文法，它所定义的语法范畴是完全独立于这种范畴可能出现的环境。

这里有许多重要的概念需要深入理解。

（1）上下文无关文法严格的定义

形式上说，一个上下文无关文法 G 是一个四元式($V_T$，$V_N$，S，P)，其中

$V_T$ 是一个非空有限集，它的每个元素称为终结符号；

$V_N$ 是一个非空有限集，它的每个元素称为非终结符号，且 $V_T\cap V_N=\varnothing$

S 是一个非终结符号，称为开始符号，$S\in V_N$

P 是一个产生式集合(有限)，每个产生式形式为 $P\to\alpha$，　其中 $P\in V_N$，$\alpha\in(V_T\cup V_N)^*$

开始符 S 至少必须在某个产生式的左部出现一次。

（2）关于文法应用的若干基本概念

严格地说，我们称 αAβ 直接推出 αγβ，即

$$\alpha A\beta\Rightarrow\alpha\gamma\beta$$

仅当 $A\to\gamma$ 是一个产生式，且 α、$\beta\in(V_T\cup V_N)^*$ 。如果 $\alpha_1\Rightarrow\alpha_2\Rightarrow\ldots\Rightarrow\alpha_n$，则我们称这个序列是从 $\alpha_1$ 到 $\alpha_n$ 的一个推导。若存在一个从 $\alpha_1$ 到 $\alpha_n$ 的推导，则

称 $\alpha_1$ 可以推导出 $\alpha_n$ 。

我们用 $\alpha_1 \Rightarrow \alpha_n$ 表示从 $\alpha_1$ 出发，经一步或若干步，可推导出 $\alpha_n$. $\alpha_1 \Rightarrow \alpha_n$ 表示从 $\alpha_1$ 出发，经 0 步或若干步，可推导出 $\alpha_n$.。换言之，$\alpha \Rightarrow \beta$ 意味着，或者 $\alpha=\beta$ 或者 $\alpha \Rightarrow \beta$。

对文法 G(E)： $E \rightarrow i \mid E+E \mid E*E \mid (E)$

$E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (i+E) \Rightarrow (i+i)$

假定 G 是一个文法，S 是它的开始符号。如果 $S \Rightarrow \alpha$，则 $\alpha$ 称是一个句型。仅含终结符号的句型是一个句子。文法 G 所产生的句子的全体是一个语言，将它记为 L(G)。

$$L（G）=\{\alpha \mid S \Rightarrow \alpha \& \alpha \in V_T^*\}$$

从一个句型到另一个句型的推导往往不唯一。

$E+E \Rightarrow i+E \Rightarrow i+i$　　$E+E \Rightarrow E+i \Rightarrow i+i$

最左推导：任何一步 $\alpha \Rightarrow \beta$ 都是对 $\alpha$ 中的最左非终结符进行替换。

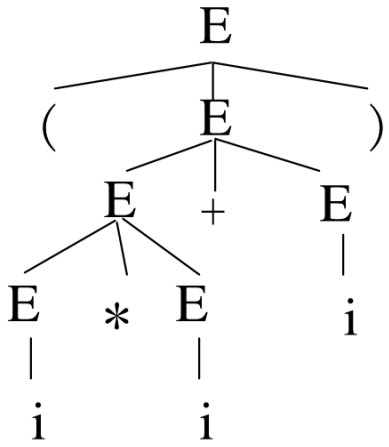
最右推导：任何一步 $\alpha \Rightarrow \beta$ 都是对 $\alpha$ 中的最右非终结符进行替换。

（3）语法分析树与二义性的概念

我们可以用一张图表示一个句型的推导,这种表示称为语法树。对于文法

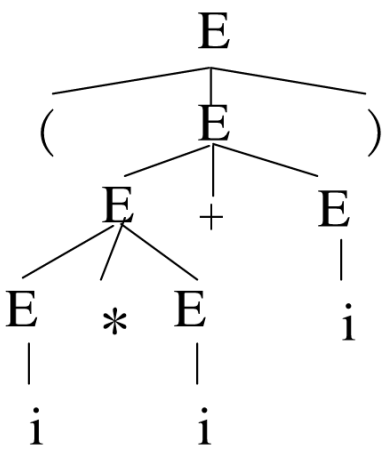
G(E)： $E \rightarrow i \mid E+E \mid E*E \mid (E)$

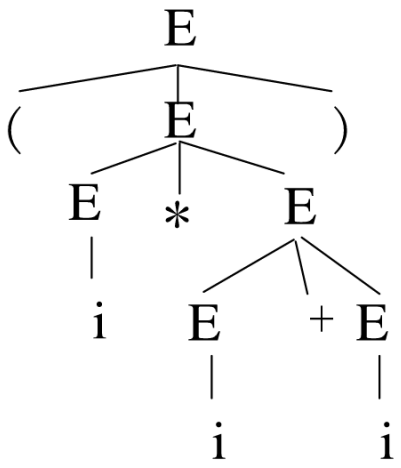
(i*i+i)的语法树如下图所示




**E ⇒(E)**
**⇒(E+E)**
**⇒(E*E+E)**
**⇒(i*E+E)**
**⇒(i*i+E)**
**⇒(i*i+i)**

一棵语法树是一些不同推导过程的共性抽象。如果使用最左(右)推导，则一个最左(右)推导与语法树一一对应。在文法中一个句型不一定只对应唯一一棵语法树。



如果一个文法存在某个句子对应两颗不同的语法树，则说这个文法是二义的。例如上面的文法 G(E)： $E \rightarrow i|E+E|E*E|(E)$ 是二义文法。需要特别指出的是，文法的二义性和语言的二义性是两个不同的概念。一个语言是二义性的，如果这个语言不存在无二义性的文法。可见，语言的二义性是一个更强的概念。对于某个语言 L 来说，可能存在两个文法 G 和 G'，其中一个为二义的，另一个为无二义的。但 L(G)=L(G')=L，这时，语言 L 不是二义性的。可以证明二义性问题是不可判定问题，即不存在一个算法，它能在有限步骤内，确切地判定一个文法是否是二义的。但是我们可以找到一组无二义文法的充分条件。

二义文法：

G(E)： $E \rightarrow i|E+E|E*E|(E)$

无二义文法：

G(E)： $E \rightarrow T \mid E+T$

$T \rightarrow F \mid T*F$

$F \rightarrow (E) \mid i$

7、乔姆斯基（Chomsky）形式语言体系

乔姆斯基于 1956 年建立了形式语言体系，为语法的描述和分析提供了坚实的理论基础。乔姆斯基把文法分成 4 种类型，即 0 型、1 型、2 型和 3 型。就文法的描述能力来说，0 型强于 1 型，1 型强于 2 型，2 型强于 3 型。这几类文法的差别在于对产生式施加不同的限制。

我们说 $G=(V_T, V_N, S, P)$是一个 0 型文法，如果它的每个产生式

$$\alpha \Rightarrow \beta$$

是这样的一种结构：$\alpha \in (V_T \cup V_N)^*$且至少含有一个非终结符，而 $\beta \in (V_T \cup V_N)^*$。0 型文法也称短语文法，其能力相当于图灵（Turing）机。或者说，任何 0 型语言都是递归可以枚举的；反之，递归可枚举集必定是一个 0 型语言。

1 型文法也称上下文有关文法。该类文法对任何产生式 $\alpha \rightarrow \beta$ 均满足$|\alpha| \leq |\beta|$（其中$|\alpha|$ 和$|\beta|$分别为 $\alpha$ 和 $\beta$ 的长度）；仅 $S \rightarrow \varepsilon$ 例外，但 S 不得出现在任何产生式的右部。这种文法意味着，对非终结符号进行替换时务必考虑上下文（上下文相关），并且，一般不允许替换成空串 $\varepsilon$。例如，假若 $\alpha A \beta \Rightarrow \alpha \gamma \beta$ 是 1 型文法 G 的一个产生式，$\alpha$ 和 $\beta$ 都不空，则非终结符 A 只有在 $\alpha$ 和 $\beta$ 这样的一个上下文环境中才可以把它替换为 $\gamma$。

2 型文法也称上下文无关文法，对应非确定下推自动机。文法中任何产生式 $A \rightarrow \beta$ ，$A \in V_N$，$\beta \in (V_T \cup V_N)^*$。从产生式的形式可以看出，非终结符的替换时不考虑上下文的。因此，语法分析时，我们通常使用下推表（先进后出存区或栈）的有限自动机来分析上下文无关语言。

3 型文法也称正规文法。3 型文法有两种形式，一种称为右线性文法，其产生式形如

$A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$，其中 $\alpha \in V_T^*$，$A、B \in V_N$。另一种称为左线性文法，其产生式形如 $A \rightarrow B\alpha$ 或 $A \rightarrow \alpha$，其中 $\alpha \in V_T^*$，$A、B \in V_N$。3 型文法等价于正规式，所以也称正规文法。

8、文法的简化

对于作为描述语言的上下文无关文法，我们对它有检点小小的限制。

（1）文法中不含任何下面形式的产生式

$$P \rightarrow P$$

因为，这种产生式除了引起二义性外没有任何用处。

（2）每个终结符 P 必须都有用处。这一方面意味着，必须存在含 P 的句型；也就是，从开始符号 S 出发，存在推导

$$S \Rightarrow \alpha P \beta$$

另一方面意味着，必须存在终结符串 $\gamma \in V_T^*$,使得 $P \Rightarrow \gamma$；也就是，对于 P 不存在永不终结回路。

我们以后所讨论的文法均家丁满足上述两条件。这种文法亦称化简了的文法。

由于同一语言可以用不同的文法来描述，显然应当选择产生式的个数最少，最符合语言特征的来描述。在文法中，有些产生式对推导不起作用，要删除掉。

简化步骤：

（1）查找有无形如 $P \rightarrow P$ 的产生式，若有则删除；

（2）若某个产生式在推导过程中永远不会被用到的，删除掉；

（3）若某个产生式在推导过程中不能从中推导出终结符，删除掉；

（4）最后，整理所有剩余的产生式，就得到简化的文法。

9、对于两个不同的文法 G 和 $G_1$，若它们的语言集相同即 $L(G)=L(G_1)$，则称 G 和 $G_1$ 等价。

例 1、对于下面程序段

```
•••
 Var      i:integer;
          a:array[1..2] of integer;
Procedure Q(b)
     Var      b:integer;
     Begin
          i:=1;b:=b+2;
          I:=2;b:=b*b
     End
Begin
   a[1]=5;a[2]:=6;i:=1;
   Q(a[i]);Q(a[i]);
Print (a[1],a[2])
End
```

若参数传递的方法分别为（1）传值，（2）传地址。请分别写出程序执行的输出结果。

解：（1）传值时输出结果为 5，,6；

（2）传地址时输出结果为 49,64。

例 2、 写一个文法使其语言为 L（G）={$a^n b^m$|2n>m≥n≥1}。

解：所求文法为 G（S）

S→ aSb|aSbb |ab

例3、 将文法 G（S）改写为等价的正规文法。

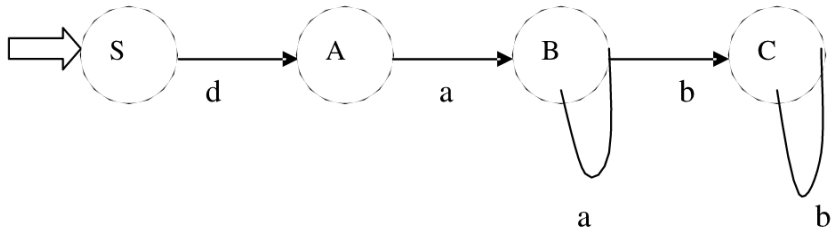
G（S）：

S→dAB

A→aA|a

B→Bb|ε

解：该文法的描述语言为 $da^i b^j$(i>0, j≥0),对应的 DFA 如图所示




相应的正规文法为：

G(S)：S→dA

A→aB

B→aB |bC|ε

C→Bc |ε

例 4、有文法 G（S）：

S→Aa |a|bC

A→aS |bB

B→Ac |bA|b

C→aB |bS

_____为 L（G）中的句子。

A、$a^{100}b^{50}ab^{100}$　　B、$a^{1000}b^{500}aba$

C、$a^{500}b^{60}aab^2a$　　D、$a^{100}b^{40}ab^{10}aa$

解：C、D

例 5、按指定类型，给出语言的文法

（a）L={$a^i b^j$|j>i≧1}的上下文无关文法。

（b）字母表上的同时只有奇数个 a 和奇数个 b 的所有串的集合的正规文法。

（c）有相同个数的 a 和 b 组成的句子的无二义文法。

解：（a）所求的文法 G1（S）：

S→AB

A→aAb|ab

B→bB|b

（b）所求的文法 G2（S）：

S→aC|bB

A→bC|ab|ε

B→aA|bS

C→aS|bA

(C) 所求的文法 G3（S）：

S→aBS|bAS|aB|bA

B→aBB|b

例 6、文法 G 的产生式集{S→S+S|S*S|i|（S）},对于输入串 i+i*i:

(1)给出一个推导；

(2) 画出一棵语法树；

(3) 文法 G 是否是二义性的，请证明你的结论。

解：(1)S⇒S+S⇒i+S⇒i+S*S⇒i+i*S⇒ i+i*i

(2) i+i*i 的语法树如图 a 所示

（3）该文法是二义性的。考虑句子 i+i*i，除了图 a 的语法树外，还有另一种语法树如图 b 所示，所以该文法是二义性的。

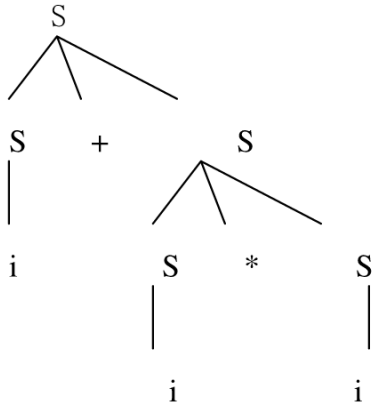



图 a

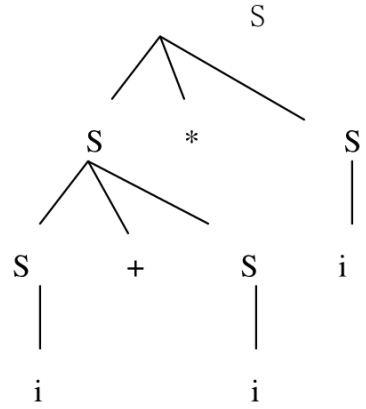



图 b

例 7、一直文法 G（S）

S→ABS|AB

AB→BA

BA→AB

A→0

B→1

(1)该文法是几型的？

(2) 该文法所产生的语言是什么？（用自然语言描述）

(3) 写出与该文法等价的 CFG 文法。

解：（1）从文法的规则上可以看出该文法是 Chomsky 1 型文法,即上下文有关文法；

(2)它生成的语言是由 1 和 0 组成的，并且是 1 和 0 的数目形同的字符串；

（3）等价的 CFG 是 G（S）：

S→0S1|1S0|01S|10S|S01|S10|SS|ε

例 8、写一文法，使其语言是偶正整数的集合,要求：

(1)允许 0 打头；

(2) 不允许 0 打头。

解：

(1)G(S)=({S,P,D,N},{0,1,2,…,9},P,S)

P:

S->PD|D

P->NP|N

D->0|2|4|6|8

N->0|1|2|3|4|5|6|7|8|9

(2)G(S)=({S,P,R,D,N,Q },{0,1,2,…,9},P,S)

P:

S->PD|P0|D

P->NR|N

R->QR|Q

D->2|4|6|8

N->1|2|3|4|5|6|7|8|9

Q->0|1|2|3|4|5|6|7|8|9

例 9、写一个上下文无关文法，使其语言是能被 5 整除且不以 0 开头的无符号整数的集合。（如{5，10，15，....}）

解：

能被 5 整除的数从形式上看，是以 0,5 结尾的数字串。题目要求的不以 0 开头，并要注意 0 不是该语言的句子。所求文法为：

G(S):

S→M F|5

F→5|0

N→1|2|3|4|5|6|7|8|9

D→N|0

M→M D| N

其中，S 代表能被 5 整除且不以 0 开头的无符号整数；F 代表可以出现在个位上的数字；D 代表所有数字；N 代表所有非零数字；M 代表所有不已零开头的数字串。