第一章

1. 程序设计语言是人与计算机联系的工具，通过程序设计语言指挥计算机按照自己的意志进行运算和操作显示信息和输出运算结果。
2. 最早的计算机程序设计语言是机器语言(指令系统)。机器语言中的指令都是用二进制代码直接表示的。
3. 机器语言和符号语言以及汇编语言都是低级程序设计语言。
4. 1954 年 FORTRAN I 语言的问世标志计算机高级程序设计语言的诞生。
5. 计算机高级程序设计语言独立于机器，比较接近于自然语言，容易学习掌握,编写程序效率高,编写的程序易读易理解易移植。
6. **翻译程序**：将高级语言编写的程序翻译成机器语言。
7. **编译程序的工作过程**：编译程序这要功能是将源程序翻译成等价的目标程序，这个翻译过程分为词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成。
8. **编译程序的重要意义**在于它使高级语言独立于机器语言，使程序员用高级语言编写程序时不必考虑那些直接与机器有关的琐碎的环节，这些细节由编译程序区处理。
9. **编译程序包括**：词法分析程序、语法分析程序、语义分析程序、中间代码生成程序、代码优化程序和目标代码生成程序以及表格处理程序和出错处理程序。
10. **编译程序的组织方式**：编译过程分为六个阶段,改划分是编译程序的逻辑组织方式。编译过程分为前端和后端。前端包括词法分析、语法分析、语义分析、中间代码生成、代码优化。后端包括目标代码生成,依赖于计算机的硬件系统和机器指令系统。这种组织方式便于编译程序的移植，如果移植到不同类型的机器上只需修改编译程序的后端即可。
11. **翻译方式**：编译方式和解释方式。
12. **源程序**：用高级语言编写的程序。源程序是编译程序加工的对象。
13. **编译方式**：先将源程序翻译成汇编语言程序或机器语言程序（目标程序），然后再执行。这个翻译程序为编译程序.
14. 编译方式中源程序的编译和目标程序的运行时分成两个阶段完成的。编译所的目标程序计算机暂时不能执行,必须由连接装配程序将目标程序和编译程序及系统子程序连接成一个可执行程序,这个可执行程序可直接被计算机执行。例如 FORTRAN,ALGOL,PASCAL,C,C++等等。
15. **解释方式**：对源程序边翻译边执行，按解释方式进行翻译的翻译程序为解释程序。优点在于便于对源程序调试和修改，加工处理过程慢。
16. **解释程序**：按解释方式进行翻译的翻译程序.
17. **词法分析**：词法分析是编译过程的基础，任务是扫描源程序，根据语言的词法规则分解和识别出每个单词，并把单词翻译成相应的机内表示。在识别单词的过程中同时也做词法检查。
18. **语法分析**：语法分析师在词法分析的基础上进行的。任务是根据语言的语法规则把单词符号串分解成格内语法单位，如表达式、语句等。通过语法分析确定整个输入符号串是否构成一个语法正确的程序。
19. **语义分析**：任务是对源程序进行语义检查，其目的是保证标识符和常数的正确使用，把必要的信息收集保存到符表或中间代码程序中，并进行相应的处理。
20. **中间代码生成**：是必不可少的阶段，任务是在语法分析和语义分析基础上把语法成分的语义对其继续翻译，翻译的结果是某种中间代码形式，这种中间代码的结果简单，接近计算机的指令形式，能够很容易被翻译成计算机指令，常用的中间代码有三元式，四元式和逆波兰式。
21. **目标代码生成**：任务:将中间代码或优化之后的中间代码转换为等价的目标代码,即机器指

令或汇编语言。由中间代码很容易生成目标程序（地址指令序列）。这部分工作与机器关系密切 ，所以要根据机器进行。在做这部分工作时（要注意充分利用累加器），也可以进行优化处理。

22.**编译程序的自展、移植与自动化**：高级语言的自编译性是指可以用这个语言来编写自己的编译程序。对于具有自编译性的高级语言，可运用自展技术构造其编译程序。即先对语言的核心部分构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。

23.**高级语言的自编译性**：是指可以用这个语言来编写自己的编译程序。一个具有自编译性的高级语言该机其他高级语言的编译程序。

24.**编译程序的移植**：将一个机器(宿主机)上的具有自编译性的高级语言编译程序移植到另一个机器上(目标机)。

25.**编译程序的自动化**：在编译程序自动化中开发早和应用广泛的是分析程序生成器和语法分析程序生成器。LEX 是一个有代表的性的词法分析生成器。输入的是正规表达式,输出的是词法分析器。LEX 的基本思想是由正规表达式构造有穷自动机。YACC 是一种基于 LALR(1)文法的语法分析生成器。他接受 LALR(1)文法生成一个相应的 LALR(1)分析表以及一个 LALR(1)分析器，而且 YACC 得语法分析程序可以和扫描器连接。在 YACC 源程序中除 2 型语言的规则之外，还可以包括一段语义程序指定相应的语义操作(填写,查找符号表,语义检查,生成语法树,代码生成)。LEX 和 YACC 是关于编译程序前端的生成器。编译程序后端(代码生成,代码优化)。

26.**编译程序编写系统(TWS)**:将有助于减轻编写翻译程序(编译程序汇编程序解释程序)工作的任何软件系统或工具包统称为翻译程序编写系统。包括产生式语言的编译程序和自动分析算法的改构造程序,以及翻译程序所必须执行的各种基本操作(建立,查找符号表操作,生成目标代码,处错处理操作等)。

27.**TWS 分为三类**：

第一类为自动产生编译程序的“编译程序的编译程序”,只要给出一个高级语言的语法规则和语义描述这类程序就能自动产生相应语言的编译程序。第二类为面向语法的符号加工程序。比较通用，例如表达式符号化简,数据格式转换,高级语言之间的相互转换等。当输入对象可用巴科斯范式 BNF 表示法加以描述时该 TWS 适用。第三类为由可扩充语言组成的集合。允许程序员用已有的数据类型和语句自定义新的数据类型和语句。

28.**规格说明**：以某种方式告知计算机所需要的是什么样的程序，要求这一程序干什么。

29.**目标语言**：是自动程序设计系统用以表示最后生成的程序的语言。

30.**问题范围**：是指希望生成的程序的应用范围，问题范围与规格说明有关，对系统采用的方法有很大影响。

31.**采用方法**：程序转换，知识工程等。

32.**串行编译程序**：适合于 SISD 结构计算机的编译程序。

33.**并行编译程序**：适合于 SIMD 和 MIMD 结构计算机，具有并行处理功能的编译程序。

34.**并行编译程序的功能**：把串行的源程序和尚未充分并行化的并行源程序自动转换成并行计算机上运行的并行目标程序或它所能接受的并行源程序。

35.**并行编译程序的任务**：实现对并行语言的翻译，受到并行语言的约束和并行计算机体系结构以及和操作系统提供的基本手段的限制。

36.并行语言分为扩充式语言和全新设计语言。

37.扩充式语言流行原因：

向上兼容串行语言。用扩充式语言编译程序可以使串行源程序不必修改或者极少的修改就可以转换成并行程序。当前全新式并行语言不能够全面的支持并行语言。兼容性不高，

不容易掌握。

38.**实现扩充语言编译程序的方式有：**

直接法：直接接受扩充式语言，并按语言的语义规则处理。

间接法：接受串行源程序(或带并行指示标志的串行源程序)，并行编译程序对源程序进行并行性检查，将检测到的并行成分转换成并行语句。或者立即进行并行编译处理。

39.**并行粒度**是对并行执行任务或者事务大小的度量。分为作业级,用户级,程序级,指令级(语句级)。作业级粒度最大，指令级粒度最小。并行编译程序应该选择适当的并行粒度。

40.**加速比** Sp 可认为是应用程序在单处理机上串行执行时间 Ts 和 p 个处理器并行执行的时间 Tp 之比，即 Sp=Ts/Tp。分析比较并行编译程序所生成的目标程序的执行速度是可用此指标。

41.**并行硬件上实现神经模型和连接机制模型途径：**

用大量的专门的神经元器件连接成特定的模型。用通用并行计算机支持各种连接模型。

第二章

1. **字母表**：字母表是元素的非空有穷集合。字母表中的元素称为符号。
2. **符号串**：符号的有穷序列成为符号串。什么符号也不包含的符号称为空符号串。符号串中符号的个数称为符号的长度。
3. **符号串相等** 若 xy 是集合上的两个符号串。且符号串的每个元素和元素的位置均相等时符号串相等。
4. 符号串的正闭包：A+ 为集合 A 上所有符号串的集合。
5. 符号串的自反闭包：A* 自反闭包不包含 A 本身 A+=AA*=A*A
6. **文法**：文法是对语言结构的定义与描述。即从形式上用于描述和规定语言结构的称为“文法”（或称为“语法”）。对于 we 妇女发 要研究它的句型、句子和语言。
7. **语法规则**：我们通过建立一组规则，来描述句子的语法结构。规定用“::=”表示“由……组成”或“定义为……”。
8. **产生式的定义**；设 VN、 VT 分别是非空有限的非终结符号集和终结符号集，V=VN∪VT ,VN∩VT=Φ 。 一个产生式是一个有序偶对（α ,β ）,其中α ∈V+,β ∈V*，通常表示为α →β 或α ::=β 。称α 为产生式的左部，称β 为产生式的右部。产生式又称为重写规则，它意味着能将一个符号串用另一个符号串替换。
9. 文法的定义：文法 G =（VN，VT，P，S）。VN：非终结符号集。VT：终结符号集。P：产生式或规则的集合。S：开始符号（识别符号） S∈VN.
10. 文法和语言分类 Chomsky 将**文法分为四类**：0 型、1 型、2 型、3 型。这几类文法的差别在于对产生式施加不同的限制。

11.**0 型文法**：P： α ::=β

其中α ∈(VN∪VT)＋，β ∈(VN∪VT)*

0 型文法称为短语结构文法。规则的左部和右部都可

以是符号串，一个短语可以产生另一个短语。

0 型语言：L0 这种语言可以用图灵机(Turing)接受。

12.**1 型文法**：

P： γ 1Aγ 2::= γ 1δ γ 2

其中 γ 1,γ 2∈(VN∪VT)*, A∈VN,

δ ∈ (VN∪VT)+

称为上下文有关文法或上下文敏感文法。也即只有在

γ 1,γ 2 这样的上下文中才能把 A 改写为δ

1 型语言：L1 这种语言可以由一种线性界限自动机接受.

13.**2 型文法**：

P： A::= δ

其中 A∈VN，

δ ∈ (VN∪VT)+

称为上下文无关文法。也即把 A 改写为δ 时，不必考虑上下文。

注意：2 型文法与 BNF 表示相等价。

2 型语言：L2 这种语言可以由下推自动机接受.

14.**3 型文法**：

（右线性） P： A::=a 或 A::=aB

其中 A、B∈VN。a∈VT。

3 型文法称为正则文法。它是对 2 型文法进行进一步限制。

3 型语言：L3 又称正则语言、正则集合这种语言可以由有穷自动机接受.

根据上述分析，L0 L1 L2 L3

0 型文法可以产生 L0、L1、L2、L3，但 2 型文法只能产生 L2，不能产生 L1。

从 0 型文法到 3 型文法，各文法描述语言的能力依次减弱。

15.**直接推导**

设文法 G＝(VN, VT, P, S)(α，β)∈P ，γ ,δ ∈ (VN∪VT)*则称符号串γ β δ 为符号串γ α δ 应用产生式α ::=β 所得到的直接推导，记为γ α δ =>γ β δ

特别有，当γ ＝δ ＝ε 时，有α =>β 即每个产生式得右部都是它左部的直接推导。

16.**最左直接推导**

在 xUy=》xuy 直接推导中，若 x 属于 Vt*，U 属于 Vn，即 U 是符号串最左非终结符，即最左直接推导，若每一步都是最左直接推导，称为最左推导。

17.**最右直接推导**

在 xUy=》xuy 直接推导中，若 y 属于 Vt*，U 属于 Vn，即 U 是符号串最右非终结符，即最右直接推导，若每一步都是最右直接推导（规范直接推导），称为最右推导（规范推导）。

18.**句型**： S 是文法 G 的识别符号，若 S*=>u，则称 u 为文法 G 的句型。S 也是。

19.**句子**： S 是文法 G 的识别符号，若 S*=>u，u 属于 Vt*，则 U 为文法 G 的句子。

20.**语言**： S 是文法 G 的识别符号,G 语言 L(G)={u|S*=>u，u 属于 V*t}，即文法的语言是文法所有句子构成的集合。

21.**语法树**：句子结构的图示表示法，它是一种有向图，由结点和有向边组成。

（1）：树中每个结点都有一个标记，该标记是 Vn 并上 Vt 并上空中的一个符号。

（2）：树的根节结标记是文法的标志符号 S

（3）：若书的一个节点至少有一个叶子结点，则该结点的标记一点是一个非终结

22 **二义性**：若对于一个文法的某一句子存在两棵不同的语法树，则该文法是二义性文法，否则是无二义性文法。（包含二义性的句子）

23.**无用非终结符号**：如果文法的某个非终结符不出现的任何一个句型中，并且不能从它推导出非终结号串，则称该非终结符称为无用非终结符号。

24.**不可达文法符号**：如果一个非终结符不出现在 we 妇女发的任何一条产生式的右部，则称该非终结符为不可达文法符号。

25. **自上而下分析方法**：从文法的识别符号出发，看是否能推导出待检查的符号串，如果能推导出这个符号串，则表明此符号串是该文法的一个句型或句子，否则便不是。或者说，以文法的识别符号作为根节点，看其是否能构造一个语法树，而且此语法树所有叶子节点从左到右所构成的符号串恰好是待检查的符号串。如果能生成这样的语法树，则表明待检查的符号串是该文法的一个句型或句子，否则便不是。

26：带回溯的自上而下分析方法：又称为不确定的自上而下分析方法。这种方法显然花费时间多，效率低。

27：确定的自上而下分析方法：是指在分析的过程中，选择某个非终结符的产生式，是根据待检查符号串的当前符号以及各产生式右部首符号而进行的，因此是确定的。

28.自下而上分析方法：基本思想：从待检查的符号串出发，看最终是否能归约（推导的逆过程）到文法的识别符号。如果能归约到文法的识别符号，则表明此待检查的符号串是该文法的一个句型或句子，否则便不是。从待检查符号串出发，在其中寻找一个称为句柄的子串，此句柄如果与文法中的某一产生式右部相匹配，那么就用此产生式左部（一个非终结符）去替换待检查符号串中的句柄，替换之后得一新符号串，然后对这新符号串作同样处理。这便是一个归约的过程。

29.文法分类：为 0 型、1 型、2 型、3 型文法。程序设计语言的语法规则属于 3 型文法（正规文法）。程序设计语言的语法和语义部分，一般属于 1 型文法（上下文有关文法），但实际上都是采用二型文法（上下文无关文法）。

30.语法分析：语法分析方法有两类，一类是自上而下分析法，另一类是自下而上分析法。为了语法分析，需讲文法的产生式存储在计算机，可以为文法建立一个产生表，为了方便还可建立一个目录表。

第三章

1. 有穷自动机(FA)分为确定有穷自动机(DFA)和非确定有穷自动机(NFA)。
2. **一个确定的有穷自动机 DFA** 是一个五元组 DFA=(Q，Σ，t，q_0，F)其中，Q 是非空有穷状态集，Σ是有穷输入字母表，t 是一个映射 Q*Σ→Q，q_0∈Q 是开始状态，F 包含于 Q 是非空终止状态集。
3. 一个状态转换图实际上是一个有穷自动机。
4. 一个**不有穷自动机 NFA** 是一个五元组 DFA=(Q，Σ，t，Q_0，F)其中，Q 是非空有穷状态集，Σ是有穷输入字母表，映射 t 为 Q*Σ→Q 的子集所成的集合(即 t 是一个多值映射)，Q_0∈Q 是开始状态，F 包含于 Q 是终止状态集。
5. **非确定有穷自动机与确定有穷自动机的主要区别有二**：一是 NFA 有一个开始状态集，而 DFA 只有一个开始状态；二是 NFA 的映射是 Q*Σ→Q 的子集所成的集合，是一个多值映射，而 DFA 的映射 Q*Σ→Q，是一个单值映射。
6. 如果自动机的弧上允许标记ε ，则称此自动机为**ε 自动机**，记为ε NDFA 或ε DFA。
7. ε 自动机的状态转换图中会出现由若干条ε 弧组成的空移环路或空移。
8. **NDFA 确定化方法**

子集法

设 NDFA A=(Σ,Q ,t ,Q0,F)是一个非确定的有穷自动机，那么可以通过子集法构造一个和它等价的确定有穷自动机 DFA A'=(Σ,Q', t', q0, F')。构造方法如下：

⑴输入字母表**Σ**不变

⑵把 NDFA A 的每一个状态子集都作为 DFA A'的一个**状态**

⑶ DFA A'的**映射**定义 t'({r1,r2,…,rm} , a) = q' = t (r1,a)∪t (r2,a)∪…∪t (rm,a)

⑷ DFA A'的**开始状态** q0＝{s1 , s2 , … , sk}，其中，si∈Q0(i =1,2,…,k)

⑸ DFA A'的**终态集**为包含原终止状态的子集组成

造表法

造表法中 DFA A'的输入字母表Σ 、开始状态 q0 和终态集 F'的确定都与子集法的构造方法一致。

状态集 Q'与映射函数 t'则是随着造表的过程而动态生成。

首先求出开始状态 q0 在接受各输入字母后状态变迁的结果，将其填入表中。然后把得到

的结果作为新的状态，再求其接受各输入字母后状态变迁的结果，填入表中，如此过程不断重复，直到最后无新结果（状态）出现为止，此时造表结束。

9. 造表法:在子集法中，如果 NFA 的状态个数 n 比较大，那么，确定化后的 DFA 的状态个数 2^n-1 将更大，其中不少状态时不可达状态。
10. I 包含 Q，状态子集 I 的ε -闭包，记为ε -closuer(I)，定义如下：
    ① 若 q∈I，则 q∈ε -closuer(I)；
    ② 若 q∈ε -closuer(I)，q'是由 q 出发经多条ε 弧所到达的状态，则 q'∈ε -closuer(I)。显然ε -closuer(I)包含 Q。
11. I 包含 Q，a∈Σ，映射 t(I,a)={q'|t(q, a= q', q∈I)=J 包含 Q I_a=ε -closuer(J)
12. **不可达状态**:动机中，从开始状态出发没有任何一条路径能达到的状态称为不可达状态。
13. **DFA 的化简**

    对于任一个 DFA，存在一个唯一的状态最少的等价的 DFA

    一个有穷自动机是化简的 ⇔ 它没有多余状态并且它的状态中没有两个是互相等价的。

    一个有穷自动机可以通过消除多余状态和合并等价状态而转换成一个最小的与之等价的有穷自动机

    “分割法”：把一个 DFA(不含多余状态)的状态分割成一些不相关的子集，使得任何不同的两个子集状态都是可区别的，而同一个子集中的任何状态都是等价的.
14. **有穷自动机的多余状态**：从该自动机的开始状态出发,任何输入串也不能到达那个状态
15. 等价状态状态 s 和 t 的等价条件是:

    1)一致性条件：状态 s 和 t 必须同时为可接受状态或不接受状态.

    2)蔓延性条件：对于所有输入符号,状态 s 和 t 必须转换到等价的状态里.
16. 正则文法 G⇒ 有穷自动机 A

    ①令正规文法 G 的终结符号集作为有穷自动机 A 的字母表。

    ②文法 G 的每一个非终结符都作为自动机 A 的一个状态，特别是文法 G 的开始符号作为自动机 A 的开始状态。

    ③在自动机 A 中增加一个新状态 Z 作为自动机的终止状态。

    ④对于文法 G 的形如 U→aV 的产生式，在自动机 A 中构造形如 t(U,a)=V 的映射。

    ⑤对于文法 G 的形如 U→a 的产生式，在自动机 A 中构造形如 t(U,a)=Z 的映射。
17. 有穷自动机 A⇒ 正则文法 G

    ①自动机每一个状态的标记，均作为正规文法的非终结符，其中，自动机开始状态的标记作为正规文法的开始符号。自动机的输入字母表中的所有符号，作为正规文法的终结符。

    ②对于自动机的映射 t（U,a）=V，构造文法的一条产生式 U→ a V

    ③对于自动机的终止状态 Z，在正规文法中增加一条产生式 Z→ε
18. 正规表达式的定义：

    有字母表 ∑, 定义在 ∑ 上的正规表达式和正规集合递归定义如下:

    1. ε 和 ϕ 都是 ∑ 上的正规表达式, 它们所表示的正规集合分别为:{ε}和 ϕ;

    2. 任何 a∈ ∑, a 是 ∑ 上的正规表达式,它所表示的正规集合为:{a};

    3. 假定 U 和 V 都是 ∑ 上的正则表达式, 它们所表示的正则集合分别记为 L(U)和 L(V), 那么 U|V, U•V 和 U*也都是 ∑ 上的正则表达式, 它们所表示的正则集合分别为 L(U)∪L(V)、 L(U)•L(V)和 L(U)*

    4. 任何 ∑ 上的正规表达式和正规集合均由 1、2 和 3 产生。
19. 正则表达式中的运算符：|--- 或（选择）•---- 连接 *--- 重复（） ---- 括号
20. 运算符的优先级：先*(闭包），后 •(连接), 最后 |(或)• 在正则表达式中可以省略.

21. 正则表达式的性质：

设 e1, e2 和 e3 均是某字母表上的正则表达式, 则有:

单位正则表达式: ε　　εe = eε = e

交换律:　e1 | e2 = e2 | e1

结合律:　e1|(e2|e3) = (e1|e2)|e3　e1(e2e3) = (e1e2)e3

分配律:　e1(e2|e3) = e1e2|e1e3　　(e1|e2)e3 = e1e3|e2e3

此外:　r* = (r|ε)*　　r** =r*　　(r|s)* = (r*s*)*

22. 正规文法 ⇒ 正规表达式

在转换之前，先将正规文法拓广，其产生式可以是

U→α V　或　U→α　　其中 U、V∈VN , α ∈VT*

转换规则为：

⑴产生式 U→α V , V→β ， U、V∈VN , α 、β ∈VT* ，转换为正规表达式 U=α β

⑵产生式 U→α U|β ，转换为 U=α *β

⑶产生式 U→α |β ，转换为 U=α |β

23. 正规表达式 ⇒ 正规文法

算法:

1)对任何正规式 r,选择一个非终结符 S 作为识别符号.

并产生产生式　　S→r

2)若 x,y 是正规式,对形为 A→xy 的产生式,重写为

A→xB　B→y,其中 B 为新的非终结符,B∈VN

同样:　对于　　A→x*y ⇒A→xA　A→y

A→x|y　⇒A→x　　A→y

第四章

1. **词法分析程序**：编译程序中完成词法分析任务的程序段
2. **扫描器**：此法分析程序负责对源程序进行扫描，从中识别出一个个的单词符号，因此，词法分析程序又称为扫描器
3. **预处理**：预处理包括删除无用的空格、跳格、回车和换行等编辑性字符，以及注解部分
4. **状态转换图**：状态转换图是一个有向图，仅包含有限个结点，每个结点表示一个状态，其中有一个初始结点，至少有一个终态结点，节点间弧的标记可以是输入字符或字符类
5. **机内符**：标识符的机内表示又称机内符，它包括标识符的全部信息

词法分析器与单词符号

6、**词法分析器**：编译程序中完成词法分析任务的程序段，称为**词法分析器**。词法分析器负责对源程序进行扫描，从中识别出一个个的单词符号，因此，词法分析器又称为**扫描器**。

单词符号是程序设计语言的基本语法单位和最小语义单位。

单词的种类

1. 关键字：if、then、else、while、do 等
2. 标识符：表示变量名、过程名等
3. 常数：无符号数、布尔常数、字符串常数等
4. 运算符：+、－、*、/ 等
5. 分界符：逗号、分号、括号等

词法分析程序的输出形式-----单词的内部形式

8、单词类别　单词自身值

几种常用的单词内部形式：

1、按单词种类分类

2、关键字、运算符和分界符采用一符一类

3、标识符和常数的单词值又为指示字（指针值）

预处理包括删除无用的空格、跳格、回车和换行等编辑性字符以及注解部分。

文法：1. <标识符>::= 字母 |<标识符>字母 |<标识符>数字

2. <无符号整数>::=数字 |<无符号整数>数字

3. <单字符分界符>::= : | + | * | , | (|)

4. <双单字符分界符>::= <冒号>=

5. <冒号>::= :

9、**标识符的处理**

标识符的类型在机内可用向量形式表示。

标识符赋予语义后，可以用来标识变量、数组、函数、过程等。

符号表用来存放在程序中出现的各种标识符及其语义特征。

定义性标识符、使用性标识符

第五章

1. 语法分析阶段的主要任务是对词法分析的输出结果—单词序列进行分析，识别合法的语法单位。语法分析分为**自上而下**和**自下而上**两类方法
2. **自上而下分析法**的基本思想是从文法的识别符号出发，试图推导出输入符号串；或试图为输入串从上而下构造一颗语法树。
3. 自上而下语法分析面临的问题：左递归：造成死循环 回溯现象：降低分析效率
4. 文法的左递归性是指文法具有以下形式的直接左递归：U →Ux | y

   或间接左递归： U+ =>Ux
5. 用扩展的 BNF 表示法消除左递归

   对巴科斯范式进行扩展，增加以下元符号：

   ①花括号{ }

   {x}：表示符号串 x 出现零次或多次。

   {x}：n 表示符号串 x 能重复出现的最大次数，m 表示符号串 x 能重复出现的最小次数。

   ②方括号[ ]：用来表示可选项。[x]= x 或 ε ，表示符号串 x 可出现 一次或不出现。

   ③圆括号( )：利用圆括号可提出一个非终结符的多个产生式右部的公共因子。
6. 直接改写法

   设产生式 U ::＝U x | y 直接左递归形式

   可将其改写为一个等价的非直接左递归形式

   U ::＝y U’ U’ ::＝x U’ | ε
7. **LL(K)文法**是一种自上而下的语法分析方法。它是从文法的识别符号出发，生成句子的最左推导。它从左到右扫描源程序，每次向前查看 k(k>1)个字符，便能确定当前应该选择的产生式。如果每次只向前查看一个字符，则称为 **LL(1)文法**。
8. 集合 FIRST、FOLLOW 与 SELECT 的构造

   ⑴设 X∈(VN∪VT)，FIRST(X)的构造

   ①若 X∈VT，则 FIRST(X)={X};

   ②若 X∈VN，其产生式为 X→a…，a∈VT,则 a∈FIRST(X)；

   若它有产生式为 X→ε ，则ε ∈FIRST(X)；

   ③如果它有产生式 X→Y…，Y∈VN

   则 FIRST(Y)-{ε }∈FIRST(X)；

   如果它有产生式 X→Y1Y2…YK(其中，Y1,Y2,…Yi-1 都是非终结符，且 Y1,Y2,…,Yi-1 * =>ε)，则 FIRST(Yi)-{ε }∈ FIRST(X)；

如果 Y1Y2…YK　　ε ，则ε ∈ FIRST(X)。

⑵ 设α ∈(VN∪VT)*，α ＝X1X2…Xn，FIRST(α)的构造

①若α ＝ε ， 显然 FIRST(α)={ε };

②若α ≠ε ，则 FIRST(X1)－{ε }∈FIRST(α)；

③若 X1X2…Xi-1 * =>ε ，则 FIRST(Xi) －{ε } ∈ FIRST(α);

若 X1X2…Xn * =>ε ，则ε ∈ FIRST(α)。

⑶设 U∈VN，FOLLOW(U)的构造

①若 U 是文法的识别符号，则$ ∈ FOLLOW(U);

②若有产生式 A→xUy，则 FIRST(y)－{ε } ∈ FOLLOW(U)；

③若有产生式 A→xU，或 A→xUy，y * =>ε ， 则 FOLLOW(A) ∈ FOLLOW(U)。

⑷集合 SELECT(U→α)的构造

① 当α 不可空时 SELECT(U→α)=FIRST(α)

② ②当α 可空时 SELECT(U→α)=FIRST(α)∪FOLLOW(U)

9. LL(1)分析器需要用到一个**分析表 M** 和一个**符号栈 S**。

10. 分析表 M 是进行 LL(1)分析的重要依据。分析表 M 是一个矩阵，它的元素 M[U,a]可以存放一条非终结符 U 的产生式，表明当符号栈 S 的栈顶元素非终结符 U 遇到当前输入字符(终结符)a 时，说应选择的产生式；元素 M[U,a]也可存放一个出错标志，说明符号栈 S 的栈顶元素非终结符 U 不应遇到当前输入字符(终结符)a。

11. 构造分析表 M 的算法

①对文法的每一条产生式 U::=α ，若 a∈FIRST(α ）,则 M[U,a]=‘U::=α ’；

②若ε ∈FIRST(α)，则 M[U,b]= ‘U::=ε ’，其中，b∈FOLLOW(U）；

③分析表 M 的其它元素均为出错标志 error，通常用空白表示。

12. 递归下降分析方法是一种确定的自上而下分析方法。它的基本思想是给文法的每一个非终结符均设计一个相应的子程序。由于文法的产生式往往是递归的，因而这些子程序往往也是递归的。所以，这种分析方法又称为递归子程序法

第六章

1. 自下而上分析法是从输入符号串出发，试图归约到文法的识别符号，或从下往上地为输入串构造一棵语法树。本章具体介绍自下而上分析法中的简单优先分析法和算符优先分析法自下而上分析法是一种“移进－归约”法。它用到一个符号栈 S，待检查符号串的符号逐个被“移进”S 栈，当栈顶符号串与某个产生式右部相匹配时，这个符号串被替换成（“归约”为）该产生式左部非终结符。

移进就是将一个终结符推进栈

归约就是将 0 个或多个符号从栈中弹出，根据产生式将一个非终结符压入栈

2. 对这个“可进行归约的符号串”，在不同的自下而上分析方法中有着不同的称呼，在简单优先分析方法中称之为“句柄”，而在算符优先分析方法中称之为“素短语”。

3. 短语和句柄

设 S 是文法 G 的识别符号，若

S+＝>xUy ， U*＝>α

则称子串α 是句型 xα y 相对于非终结符 U 的**短语**。其中，U∈VN ； x,y∈(VN∪VT)*

如果α 是 U 的一个直接推导，即 U=>α 则称子串α 是句型 xα y 相对于非终结符 U 的**直接短语**或**简单短语**。

4. 一个句型的最左直接短语，称为该句型的**句柄**。

5. 一个句型，有一棵相应的语法树（无二义性）。该语法树的一棵子树的叶子结点（从左到右）组成的符号串便是这个句型关于子树根结点的一个**短语**。语法树的一棵简单子树（只

有单层的子树）的叶子结点组成的符号串则是这个句型关于简单子树根结点的一个**直接短语**。语法树中最左边的简单子树叶子结点组成的符号串就是这个句型的**句柄**。

6. **移进－归约方法**

一个移进－归约分析器设置一个符号栈和一个输入缓冲区。输入符号串为α ，分析开始时，栈和输入缓冲区的格局为：

符号栈	输入符号串
#	α #

对α 进行分析时，将α 的符号从左到右逐个移进符号栈，**一旦栈顶符号串与某一个产生式右部相匹配成为一个可归约符号串时，则将此符号串归约为一个非终结符，这个非终结符是该产生式的左部符号。**α 的符号继续逐个移进符号串，重复这个过程…如果最终出现如下格局：

符号栈	输入符号串
#S	#

说明符号串α 是文法的一个句子。否则，则表示α 存在语法错误。

7. 有关文法的一些关系**关系**

在集合Σ1 到Σn 上的一个 n 元关系 R，是笛卡儿乘积Σ1×Σ2×…×Σn 的一个子集。或者说，一个 n 元关系 R 可定义为一个有序的 n 元组集合。即：设 x1∈Σ1 , x2∈Σ2 ， …，xn∈Σn ，则 x1,x2,…,xn 满足关系 R，当且仅当有序 n 元组(x1,x2,…,xn)∈R。

一般 n=2，不妨令Σ＝Σ1＝Σ2，二元关系定义如下：

集合上的二元关系 R，是Σ2 的一个子集。或者说，x1 与 x2(x1,x2∈Σ)满足二元关系 R，当且仅当有序偶对(x1,x2)∈R。x1 与 x2 满足关系 R，记为 x1Rx2。

关系具有以下基本性质

⑴**自反性**设 R 是定义在集合Σ上的一个关系，如果对于任何 x∈Σ，都有 xRx，则称关系 R 是自反的。如数学中的关系“≤”是自反的，但关系“<”不是自反的。

⑵**对称性**如果对任何 x，y ∈Σ，xRy，都有 yRx，则称关系 R 是对称的。如数学中的关系“=”是对称的。

⑶**传递性**对任何 x，y ∈Σ，如果能由 xRy 与 yRz 推得 xRz，则称关系 R 是传递的。如数学中的关系“<”是传递的。

和文法相关的一些关系：

⑴**关系和**

设 x，y ∈Σ，R1 与 R2 是上的两个关系，R1 与 R2 的关系和记为 R1+R2，x(R1+R2)y 当且仅当 xR1y 或 xR2y。关系“<”与关系“=”的关系和为<+=。如 3(<+=)5，9(<+=)9。

⑵**关系积**

关系 R1 与 R2 的关系积记为 R1R2 ，x R1R2y 当且仅当存在一个 w，使得 x R1w 且 w R2y。关系“<”与关系“<”的关系积为<<，如 12<<25，因为在 12 与 25 之间的任意一个数都满足这种关系，如存在一个数 19，使得 12<19，且 19<25。

⑶**传递闭包**

关系 R 的传递闭包记为 R+，设 x，y ∈Σ ，x R+ y 当且仅当存在一个 n>0，使得 x Rny。其中， $R^1 = R$ $R^2 = RR$ $R^3 = R^2 R=R R^2$…$R^n = R^{n-1} R=R R^{n-1}$

⑷**自反传递闭包**

关系 R 的自反传递闭包记为 R*，R*=R0+R+，其中，R0 为恒等关系。设 x，y ∈Σ；x R0y 当且仅当 x=y。

定理 1 关系 R 的转置关系记为 RT，则 M(RT)=M（R）T 即转置关系（RT）的关系矩阵等于关系矩阵的转置矩阵。

定理 2 关系 R1 与 R2 的关系和(R1+R2)的关系矩阵，等于 R1 的关系矩阵与 R2 的关系矩阵之和，即 M(R1+R2)=M(R1)+M(R2)

定理 3 关系 R1 与 R2 的关系积(R1R2)的关系矩阵，等于 R1 的关系矩阵与 R2 的关系矩阵之积，即 M(R1R2)= M(R1)M(R2)

定理 4 关系 R 的传递闭包(R+)的关系矩阵，等于 R 的关系矩阵的传递闭包，即 $M(R^+)=M（R）^+$

8. **关系 FIRST 与 LAST**

①A FIRST B，当且仅当文法有如下产生式： A::=Bα 其中，A∈VN，B∈(VN∪VT)，α ∈(VN∪VT)* 。

②A LAST B，当且仅当文法有如下产生式： A::=α B 其中，A∈VN，B∈(VN∪VT)，α ∈(VN∪VT)* 。

9. **两个重要的结论**

①A +=>Bα ，当且仅当 A FIRST+ B 其中，A∈VN，B∈(VN∪VT)，α ∈(VN∪VT)* 。

②A+=>α B，当且仅当 A LAST+ B 其中， A∈VN，B∈(VN∪VT)，α ∈(VN∪VT)*

10.**算符优先分析方法**是根据**算符（终结符）之间的优先关系**来寻找可归约串的一种自下而上语法分析方法。

11. **简单优先关系**

①**L±R**，当且仅当文法中有以下形式的产生式：U::=α LRβ 中，L,R∈(VN∪VT)；α ，β ∈(VN∪VT)*。

②**L<R**，当且仅当文法中有以下形式的产生式：U::=α LPβ 且 P+=> Rγ

其中，L,R∈(VN∪VT)；P∈VN；α ,β ,γ ∈(VN∪VT)*。

③**L>R**，当且仅当文法中有以下形式的产生式：U::=α WPβ 且 W +=>δ L , P *=> Rγ

其中，L,R∈(VN∪VT)； W∈VN； α ,β ，δ ，γ ∈(VN∪VT)*。

12. **算符优先文法：**

①a=b,且仅当文法中包含有形如 U::=…ab…，或 U::=…aVb…的产生式；

②a<b,且仅当文法中包含有形如 U::=…aV…的产生式，其中，V+=>b…或 V+=> Wb…；

③a>b，当且仅当文法中包含有形如 U::=…Vb…的产生式，其中，V+=>…a 或 V+=>…aW。

13. 部不包含两个相邻非终结符的文法称为**算符文法**，或 **OG 文法**。

例：算符文法 G[E]

E::= E+T | T　　T::= T*F | F　　F::= (E) | i

14. 文法，如果任意两个终结符之间至多存在一种算符优先关系，则称此算符文法是一个**算符优先文法**或 **OPG 文法**。

15. **构造算符优先关系：**

先定义如下两个集合：

最左终结符集：**FIRSTVT(U)**={b | U+=> b…，或 U+=>Vb…，V∈VN}

最右终结符集：**LASTVT(U)**={a | U+=>…a，或 U+=>aV，V∈VN}

①算符优先关系“=”：可直接检查产生式，若有形如 U::=…ab…，或 U::=…aVb…的产生式，则(a , b)∈ = 。

②算符优先关系“<”：若有形如 U::=…aV…的产生式，则对于 b ∈ FIRSTVT(V)，有(a , b)∈ < 。

③算符优先关系“>”：若有形如 U::=…Vb…的产生式，则对于 a ∈ LASTVT(V)，有(a , b) ∈ > 。

构造集合 FIRSTVT(U)建立一个布尔数组 F[U,a]，使得 F[U,a]为真的条件是：当且仅当 a ∈ FIRSTVT(U) 。开始时，数组元素的初值为假。观察文法，若有形如 U::=a…或 U::=Va…

产生式，则令相应的数组元素值为真，并将其存入堆栈 S 中。然后，对堆栈施行如下操作：如果栈 S 不空，就将栈顶项弹出，记此项为(Q , a)。对于每个形如 U::=Q…的产生式，若 F[U,a]为假，则将其置为真且将(U , a)存入栈 S 中。上述过程一直重复进行，直到堆栈 S 为空。

16. 素短语及句型的分析

素短语是至少包含一个终结符的短语，但它不能包含其它素短语。

例：设文法 G[E]

E::= E+T | T

T::= T*F | F

F::= (E) | i

短语： T+i*(E) , T , i*(E) , i , (E)

素短语： i , (E)

注：算符优先分析法中将**最左素短语**作为可归约符号串。

17. 由算符文法的定义，算符文法的句型可写成以下形式：

[V1]a1[V2]a2…[Vn]an[Vn+1]其中，ai∈VT , Vi∈VN(i=1,2, … , n+1)，Vi 为可选项。

18. **定理**：一个算符优先文法的句型的**最左素短语**是该句型中满足下列条件的最左子串 [Vi]ai…[Vj]aj[Vj+1]：

ai-1 < ai , ai = a i+1 , … , aj-1 = aj , aj > aj+1

19. **算符优先分析算法**

分析中用到一个符号栈 S。每一步分析均要比较栈内最顶端终结符 S[i]与当前输入符号 TR[j]之间的优先关系：

若 S[i]<TR[j]或 S[i] = TR[j]，则当前输入字符 TR[j]入栈；

若 S[i]>TR[j]，则表示栈内出现最左素短语（栈顶符 S[m]即为最左素短语尾字符），此时在符号栈内自上而下依次比较下、上两相邻（至多相隔一个非终结符）终结符之间的关系，若发现 S[k] < P ，则表示 S[k+1]为最左素短语首字符。最左素短语为 S[k+1]…S[m]，根据文法产生式，将其归约。如此步骤重复进行，直到分析过程结束。

注：归约时，要求最左素短语和相关产生式右部的终结符位置相同、符号相等，非终结符只要求对应位置相同。

20. **优先函数及其构造**

优先函数

设 M 是简单优先文法 G 的简单优先关系矩阵，符号 L 与 R 只存在一种简单优先关系，如果以文法字母表作为定义域的函数 f 和 g 满足以下条件：

①若 L=R，则 f(L) = g(R)

②若 L>R，则 f(L) > g(R)

③若 L<R，则 f(L) < g(R)

那么，f 和 g 称为对于文法 G 的优先关系矩阵 M 的**优先函数**。

注意：不是所有的优先关系矩阵都有对应的优先函数。

Bell 方法

Bell 方法是利用**有向图**来构造优先函数的一种方法。

设文法的字母表 V={A1 , A2 , … , An}，则其优先函数构造步骤如下：

①作一个具有 2n 个结点的有向图。2n 个结点通常分为上下两排，上排结点标记分别为 f1 , f2 , … , fn，下排结点标记分别为 g1 , g2 , … , gn。**如果 Ai>Aj，则从结点 fi 到结点 gj 作一条弧；如果 Ai<Aj，则从结点 gj 到结点 fi 作一条弧；如果 Ai=Aj，则既从结点 fi 到结点 gj 作一条弧，也从结点 gj 到结点 fi 作一条弧。**

②给有向图中的每一个结点赋一个数值，**这个值等于该结点所能到达的结点（包括该结点自身）的个数**。函数 f(Ai)等于结点 fi 的值，函数 g(Aj)等于结点 gj 的值。

③**检查函数 f(Ai)和 g(Aj) （i=1,2,…,n）是否与文法的优先关系矩阵一致**，即是否符合优先函数定义。如果一致，则函数 f(Ai)和 g(Aj)便是优先函数，否则它们就不是优先函数。

Floyd 方法

①对于文法的所有符号 A∈(VN∪VT)，令 f(A)=g(A)=1；

②对于 A<B，如果 f(A)≥g(B)，则令 g(B)=f(A)+1；

③对于 A>B，如果 f(A)≤g(B)，则令 f(A)=g(B) +1；

④对于 A=B，如果 f(A)≠g(B)，则令 f(A)=g(B)=max(f(A) , g(B))；

⑤重复步骤，直到 f 和 g 都符合优先函数定义为止，否则不存在与文法的优先关系矩阵相对应的优先函数。

第七章

1. LR(k)分析器是这样一种分析程序：它总是按从左至右方式扫描输入串，并按自下而上方式进行规范归约。在这种分析过程中，它至多只向前查看 k 个输入符号就能确定当前的动作是移进还是归约；若动作为归约，则它还能唯一地选中一个产生式去归约当前已识别出的句柄（这里称为活前缀）。若该输入串是给定文法的一个句子，则它总可以把这个输入串归约到文法的开始符号；否则报错，指明它不是该文法的一个句子。

2. 给定文法 G，S 是其开始符号。考虑该文法中一个终结符号串 w 的一个规范推导 S=>w1=>w2=>…=>w 假定 uAv=>uxv 是上述推导中的一个推导步。A::=x 是用于该推导步的产生式。对于每一个这样的推导和推导步，仅通过扫描 ux 和查看 v 中开始的 k 个符号就能唯一确定选用产生式 A::=x，我们就称 G 为 **LR(k)文法**。

3. LR 分析程序是一个确定的下推自动机。

4. LR 分析程序的分析表由两部分组成：**分析动作表**(ACTION 表)和 **goto 函数表**(gato 表)。

分析动作表（ACTION）：

action[sm,ai]= 移进 ai 和 s=goto[sm,ai]进栈
归约 rj : A::=Xm-r+1Xm-r+2…Xm
接收 s=goto[sm-r , A]
错误处理

GOTO 函数表：GOTO[si,xj]规定了当状态 si 面临文法符号 xj 时所应到达的下一状态。

6. **LR 分析器的逻辑结构** 一个输入、一个输出、一个栈、一个驱动程序和一张分析表。

7. **LR 分析器的工作过程**

格局（构形）：（栈中符号序列，剩余输入符号串）

开始：（s0, a1 a2 ……an$)

中间：（s0x1s1x2s2…xmsm ,aiai+1…an$)

1. 若 action[sm ,ai]=si 则把 ai,si=action[sm ,ai]推进栈。

格局：（s0x1s1x2s2…xmsm aisi , ai+1…an$)

2. 若 action[sm ,ai]=r (A::=xm-r+1xm-r+2…xm) ,则

格局：（s0x1s1x2s2…xm-rsm-r As , ai ai+1…an$)其中，s=goto[sm-r ,A]

3.若 action[sm ， $]=accept,则分析结束。

4 .若 action[sm,ai]=error,则转出错处理程序。

8. **LR 分析方法的基本原理**是：把每个句柄（某个产生式的右部）的识别过程划分为若干状态，每个状态从左至右识别句柄中的一个符号，若干个状态就可识别句柄左端的一部分符号。识别了句柄的这一部分就相当于识别了当前规范句型的左起部分——规范句型的活前缀。因而，对句柄的识别就变成了对规范句型活前缀的识别。

9. **LR(0)分析程序**，即在分析的每一步，仅根据当前的栈顶状态就能确定应执行何种分析动作，而无须向前查看任何输入符号。

10. **活前缀**（Viable Prefix）是指规范句型的一个前缀，它不包含该句型的句柄右边的任何符号

11. **LR(0)项目**：用圆点“·”表示识别一个产生式右部符号到达的位子

12. 给定文法 G，S 是其开始符号，我们构造一个与 S 相关的文法 G’：它包含整个 G，而且外加一个新产生式 S’::=S。其中，S’是 G’的开始符号，称 G’为 G 的**拓广文法**。

13. $A\rightarrow \alpha\cdot a\beta \quad a\in VT$　　移进项目

$A\rightarrow \alpha\cdot B\beta \quad B\in VN$　　待归约项目

$A\rightarrow \alpha\cdot$　　归约项目

$S'\rightarrow S\cdot$　　接收项目

14. **CLOSURE(I)函数**

设 I 是文法 G 的一个 LR(0)项目集合

closure(I)是从 I 出发用下面三个规则构造的项目集:

1．I 中每一个项目都属于 closure(I)。

2．若项目 $A\rightarrow \alpha B\beta \in$closure(I)且 $B\rightarrow \gamma \in P$

则将 $B\rightarrow \gamma$ 加进 closure(I)中。

3．重复执行(2)直到 closure(I)不再增大为止。

15. **goto(I,X)函数**

若 I 是 G 的一个 LR(0)项目集, $X\in\{VT\cup VN\}$ 则 goto(I,X)=closure(J)

其中， $J=\{A\rightarrow \alpha X\beta \mid$当 $A\rightarrow \alpha X\beta \in I$ 时$\}$

goto(I,X)称为转移函数。项目 $A\rightarrow \alpha X\beta$ 称为 $A\rightarrow \alpha X\beta$ 的后继。

16. **有效项目**

一个项目[A→x.y]称为对某个活前缀 ux 是有效的，当且仅当存在某个规范推导

S=>uAv=>uxyv 其中，xy 是规范句型 uxyv 的句柄，v 是一个终结符号串。

17. **LR(0)文法**

如果文法 G’的项目集规范族的每个项目集中不存在下述任何冲突项目：

①移进项目和归约项目并存；

②多个归约项目并存；

则称文法 G’为 **LR(0)文法**。

当且仅当一个文法是 LR(0)文法时，才能构造出它的不含冲突动作的 LR(0)分析表。

18. **构造 LR(0)分析表的算法**

设一文法 G‘的项目集规范族 C＝{I0,I1,…,In},令其中每个项目集 Ii 的下标作为分析器的状态，令包含项目 S’→.S 的项目集 Ik 的下标 k 为分析器的初态，则构造 LR(0)分析表的步骤为：

① 若项目 $A\rightarrow \alpha .x\beta \in Ii$ 且 goto(Ii,x)=Ij,$x\in\Sigma$,则置 ACTION[i,x]=Si，即“将状态 j、符号 x 移进栈”；但若 $x\in VN$，则仅置 GOTO[i,x]=j。

② 若项目 $A\rightarrow \alpha .\in Ii$，对于任何输入符号 $a\in(\Sigma\cup\{\$\})$，则置 ACTION[i,a]=rj，即“用第 j 条产生式 $A\rightarrow \alpha$ 进行归约”。

③ 若项目 S’→S.∈Ik，则置 ACTION[k,$]=“接收”。

④ 分析表中凡不能用规则①～③填入信息的元素均置上 ERROR（用空白表示）。

19. 构造 LR(0)分析表的步骤小结

①写出给定文法 G 的拓广文法 G’；

②写出文法 G’的基本 LR(0)项目——G’的 LR(0)项目集；

③利用 CLOSURE 和 GOTO 函数，求出相应的 LR(0)项目集规范族 C；

④构造识别该文法全部活前缀的 DFA；

⑤根据算法构造 LR(0)分析表。

20. 构造出的文法的 DFA 如果没有移进-归约、归约-归约冲突则该文法为 LR(0)文法。考察 **SLR(1)文法**：如果有移进-归约冲突，则考察移进-归约项目中的 FOLLOW(归约) ∩移进项目=∮ 如果有归约-归约冲突，则考察归约-归约项目中的 FOLLOW(归约 1) ∩FOLLOW(归约 2)=∮，如果有移进-归约、归约-归约冲突，则考察 FOLLOW(归约 1) ∩FOLLOW(归约 2) ∩移进项目=∮。

21. 重新定义项目，使得每个项目包含两个部分，第一部分就是原来的项目本身，第二部分是由一个终结符号（可能为$）组成。重新定义后的项目称为 **LR(1)项目**，其一般形式为[A→α .β ,a]项目中第二部分称为**向前看符号**

22. **构造规范 LR(1)分析表的算法**

设一文法 G'的 LR(1)项目集族 C＝{I0,I1,…,In},令其中每个项目集 Ii 的下标作为分析器的状态，令包含项目[S'→.S,$]的项目集 Ik 的下标 k 为分析器的初态，则构造规范 LR(1)分析表的步骤为：

①若项目[A→α .xβ ,b]∈Ii 且 goto(Ii,x)=Ij,x∈Σ,则置 ACTION[i,x]=Si，即"将状态 j、符号 x 移进栈"；但若 x∈VN，则仅置 GOTO[i,x]=j。

②**若项目[A→α .,a]∈Ii，则置 ACTION[i,a]=rj，即"用第 j 条产生式 A→α 进行归约"。**

③若项目[S'→S.,$]∈Ik，则置 ACTION[k,$]="接收"。

④分析表中凡不能用规则①～③填入信息的元素均置上 ERROR（用空白表示）。

23. **LALR（向前看 LR）分析技术**是介于规范 LR 分析器构造法和 SLR 分析器构造法之间的一种方法。某些 LR(1)项目集称为**同心**，即如果除了向前看符号之外，这些项目集是相同的。在 LALR 分析方法中，就试图将这些同心项目集合并成一个项目集。

24. **构造 LALR 分析表的算法**

设一文法 G'的 LR(1)项目集族 C＝{I0,I1,…,In},令其中每个项目集 Ii 的下标作为分析器的状态，则构造 LALR 分析表的步骤为：

①**合并 C 中的同心集**，记合并后的项目集族为 C'＝{J0,J1,…,Jm},令其中每个项目集 Ji 的下标作为分析器的状态，令包含项目[S'→.S,$]的项目集 Jk 的下标 k 为分析器的初态，

②若项目[A→α .xβ ,b]∈Ji 且 goto(Ji,x)=Jj,x∈Σ,则置 ACTION[i,x]=Si，即"将状态 j、符号 x 移进栈"；但若 x∈VN，则仅置 GOTO[i,x]=j。

③若项目[A→α .,a]∈Ji，则置 ACTION[i,a]=rj，即"用第 j 条产生式 A→α 进行归约"。

④若项目 S'→S.∈Ik，则置 ACTION[k,$]="接收"。

⑤分析表中凡不能用规则①～③填入信息的元素均置上 ERROR（用空白表示）。

第八章

1. **制导翻译的基本思想**是先给文法中的每个产生式添加一个成分，这个成分常称为语义动作或翻译子程序，在执行语法分析的同时，执行相应产生式的语义动作。

2. 所谓语法制导翻译法就是在语法分析的过程中，依从分析的过程，根据每个产生式添加的语义动作进行翻译的方法。

3. 语法制导翻译法 SDTS 是一个五元组 T=(VT,VN,△,R,S)。VT 是一个有穷的输入字母表，包含源语言中的符号；VN 是一个有穷的非终结符号集合；△是一个有穷的输出字母表，包含出现在翻译串中的那些符号；R 是形如 A→w,y 的规则的有穷集合，w 是由终结符或非终结符组成的串，y 是由 VN 或△中的符号组成的串；S∈VN 是一个开始符号。

例如：SDTS T1=

({a,b,c,+,-,[,]},{E,T,A},{ADD,SUB,NEG,x,y,z},R,E)，
其中 R 由下列翻译规则组成：

① E→E+T , T E ADD
② E→E-T , E T SUB
③ E→–T, T NEG
④ E→T , T
⑤ T→[E] , E
⑥ T→A , A
⑦ A→a , x
⑧ A→b , y
⑨ A→c , z

T1 的**基础源文法**是
E→E+T|E-T|-T|T
T→[E]|A
A→a|b|c
T1 的**基础目标文法**是
E→T E ADD|E T SUB
| T NEG |T
T→E|A
A→x|y|z
一个**翻译模式**是一个形如(u,v)的串对，其中，u 是 SDTS 基础源文法的一个句型，而 v 称为与其对应的翻译，它是由 VN 和△中元素组成的串。

4. 翻译模式的定义如下：
   ① (S,S)是一个翻译模式，且这两个 S 是相关的(S 是 SDTS 的开始符号)。
   ② (aAb,a'Ab')是一个翻译模式，且两个 A 是相关的；此外，若 A→g,g'是 R 中的一条规则，那么(agb,a'g'b')也是一个翻译模式。规则中 g 和 g'的非终结符之间的相关性也必须带进这种翻译模式之中。

   表示法　(aAb, a'Ab')=>(agb,a'g'b')
   表示一种翻译模式到另一种翻译模式的变换。
5. 树变换
   语法制导的翻译过程也可用语法树变换来说明。
   ① 从中剪掉终结符号节点；
   ② 根据适当的翻译规则，重排每个中间结点的孩子；
   ③ 添加对应于输出符号集中的中间符节点。
6. **简单 SDTS 和自上而下翻译器**
   如果在每一规则的翻译成分中，非终结符出现的次序与它们在源成分中出现的次序相同，则称一个 SDTS 是**简单的**(simple)；但是，如果 A→A1+A2 , A2　A1　ADD 是 SDTS T 的一个规则，那么，该 SDTS　T 就是不简单的。如果 T=(VN,VT,△,R,S)是其基础源文法为 LL(k)的简单 SDTS，那么，存在一个自上而下的确定的下推翻译器 PDT(Push-Down Translater)，它接受 T 的输入语言中的任何符号串并产生对应的输出串。
7. PDT　P 的定义如下：P 的一个构形是一个四元组(q,x,y,z)，其中，q 是它的有穷控制器的状态，x 是尚待扫描的输入串，y 是下推栈，z 是此时被打印出的输出符号串。于是，在某次移动中，便有(q,ax,Yy',z) ⊢(r,x,gy',zz')即，在状态 q 面临输入符号 a 且栈顶符号为 Y 时，P 才允许移动到状态 r，这一移动的结果是：输入符号 a 被去掉，栈顶符号 Y 被 g 所

替换，而且串 z’已作为输出串打印出。称 w 是关于 x 的输出，如果对于某个状态 q 和栈符号串 u，存在(q0,x,Z0,ε) ⊢(q,ε ,u,w)其中，q0 是初态；Z0 是栈的初始内容，ε 为空串。若 u＝ε ，则称 P 停止于空栈；若 q 是某个终态，则称 P 停止于终态。如果 P 满足下述两个条件，则称 P 是**确定的**：

① 对所有的状态 q，输入串 a 和栈符号 Z，δ (q,a,Z)至多只包含一个元素；

② 若δ (q,ε ,Z)非空，则不存在符号 a(a≠ε)使得δ (q,a,Z)非空，即对于某个状态和栈符号，在空移动和非空移动之间不应存在冲突。

8. PDT P 的操作为：

①在一应用移动中，非终结符 A 已位于栈顶，借助于分析表，选定产生式 A→w 来进行归约。假定 R 中的翻译规则为 A→w ，z 其中，w=a0B1a1B2a2…Bkak，而 z=b0B1b1B2…Bkbk。其中，Bi 是非终结符，ai 是输入符号串，bi 是输出符号串。位于栈顶的 A 由下面的复合串 b0a0B1b1a1B2…Bkbkak 所替代，而 b0 称为新栈顶符号。

②若栈顶符号是输出字母表的一个元素，则从栈中弹出，并将其作为输出符号打印出。

③若栈顶符号是输入字母表 VT 的一个元素，则它应与当前输入符号匹配。从栈顶弹出它，输入指针后移。

9. **简单后缀 SDTS 和自下而上翻译器**

如果一个 SDTS 是简单的，而且它的每个翻译规则都有下述形式：A→a0B1a1B2a2…Bkak，B1B2…Bkw 也就是说，除了最右边的 w 之外，（输出）终结符不可能出现在翻译成分之中，那么，称这个 SDTS 为**简单后缀的**(Simple Post Fix)。

定理：对其基础源文法为 LR(k)的每一简单后缀的 SDTS，存在一确定的 LR(k)PDT：

① 它接受从该基础源文法可推导出的每一句子；

② 它将这种句子的翻译作为输出。

10.**抽象语法树** AST(Abstract-Syntax tree)是某个语言结构的一种简洁的树形表示形式，它只需包含该结构尚须转换或归约的信息。

语法树简化成一个 AST：

① 去掉于单非产生式（即右部为单一终结符的产生式）相关的子树，并上提相关分支上的终结符号节点。

② 对于直接包含运算符的多个子树，上提运算符并让它取代其父节点。

③ 去掉括号，并上提运算符，让它取代其节点。

最后得到的语法树便是一个 AST。

11.**属性文法**是以上下文无关文法为基础，只不过为每个文法符号配备了一些属性。属性同变量一样，可以进行计算和传递。属性的加工过程即是语义处理的过程。

12.**继承属性**：其计算规则按“自上而下”方式进行，即文法产生式右部符号的某些属性根据其左部符号的属性和右部的其它符号的某些属性计算而得。

13.**综合属性**：其计算规则是按“自下而上”方式进行，即产生式左部符号的某些属性根据其右部符号的属性和自己的其它属性计算而得。

例如：　　Ap→Xq,rYs,t

　　[p=q+s，r=p+t]

其中，p、q、r、s、t 是属性。p 是综合属性，r 是继承属性

14. **L 属性文法**

①产生式右部任一文法符号 V 的继承属性仅依赖于下述两种属性值中的一种：

a) 产生式左部的继承属性；

b) 产生式右部但位于 V 左边的符号的任何属性。

②产生式左部符号的综合属性仅依赖于下面的属性值中的一种：

a) 产生式左部符号的继承属性；
b) 产生式右部符号（除自身外）的任意属性。

15.**S 属性文法**

① 全部非终结符的属性是综合属性；
② 同一产生式中相同符号的各综合属性之间无相互依赖关系；
③ 如果 q 是某个产生式中文法符号 V 的继承属性，那么，属性 q 的值仅依赖于该产生式右部位于 V 左边的符号的属性。

16.所谓**中间代码形式**是指用来表述源程序并与之等效的一种编码方式。

常见的中间代码形式有逆波兰表示、四元式、三元式和树形表示等等。

逆波兰表示法：运算符直接跟在其运算量（操作数）的后面

例如： AB* 表示 A*B
AB*C+ 表示 A*B+C
ABCD/+* 表示 A*(B+C/D)

例：

```
begin
    integer  k;
    k:=10;
  h: if k>i+j  then
        begin
            k:=k-1;
            goto  h
        end
     else
            k:=i*2-j*2;
    i:=j:=0
end
```

四元式一般形式： （运算符 运算量 1 运算量 2 结果）

例如，表达式 A*B+C*D 的四元式为

*** A B T1**
*** C D T2**
+ T1 T2 T3

三元式一般形式： （运算符 运算量 1 运算量 2 结果）

例如，A+B*C 可表示为 ① * B C
② + A ①

第九章

1. 为管理过程活动所需的信息，当过程调用发生时，使用一个连续的存储块记录该活动的相关信息，称为该过程的活动记录或数据区。
2. 当过程返回（活动即将结束）时，过程的数据区或活动记录一般包含连接数据、形式单元、局部数据。
3. 过程（或分程序）嵌套结构的程序设计寓言，源程序书写完后，过程（或分程序）间形成一种嵌套层次关系，过程（或分程序）在这种嵌套结构中的层次成为静态层次。
4. 直接包含某过程（或分程序）的外层成为该过程（或分程序）的静态外层。
5. 源程序中，定义一个标识符的语法单位成为该标识符的作用域。
6. 标识符的生存期是指定义该标识符的语法单位的整个运行期间，出去该语法单位内部嵌

套的定义有同命标识符的语法单位的运行期间。

7. 在主程序中说明的量成为全程量。对于嵌套子程序结构，在外层子程序中说明的量称为相对于原子程序的非局部量，而在内层子程序中说明的量称为相对于内层子程序的局部量。

8. 如果一个名字的属性是通过说明语句或隐约定规则定义的，则称这种性质是静态属性。如果名字的属性只有在程序运行时才能知道，则称这种属性是动态属性。

9. 数组的存储可以分为单块存储方式和多块存储方式。

10.静态存储分配，是指在编译阶段对源程序中的量分配以固定存储单元，运行时始终不变。

11.动态存储分配，是指在运行阶段动态地为源程序中的量进行存储单元分配。

12.栈式存储分配，是指在运行时把存储器作为一个栈进行管理，每当调用一个过程，它所需的存储空间分配在栈顶，一旦退出，它所占用的空间就被释放。

13.堆式存储分配，是指在运行时把存储区组织成堆，以便用户对存储空间进行申请与归还，凡申请则从堆中分给一块，凡释放则退回给堆。

14.程序运行时，调用者与被调用者之间的信息交流是通过全局量或参数传递的方式进行的。

15.以过程为单位的栈式存储分配，是指把整个程序的数据空间设计成一个栈，以过程调用为单位来设置数据区。

第十章

1. 符号表的每一项（称为表项）包含两个部分，即主目和值。

2. 表的主目通常即符号或标识符本身（变量的名字），值是它们的属性，包括种属和类型。

3. 符号表的构造（简称造表）是指把新的表项填入表中的过程。

4. 符号表的查找是致在符号表中搜索某一特定表项的过程。

5. 构造符号表是最容易的方法是线性构造表法。

6. 折半造表法是指按照表项主目的“大小”次序进行填写，“小”的排在前，“大”的排后。

7. 杂凑技术就是设计一杂凑函数 h(x)，其中 x 为任一标识符，它把 x 映像成表区中某一单元的地址，并要求当 x≠y 时，h(x)=h(y)的可能性尽量地小，即单元冲突的可能性尽量地小。

8. 线性探查法是解决冲突的最简单的一种方法，即当发生冲突时，从冲突单元的下一位置开始，逐个查寻，直至找到第一个所需位置（空单元）或指出语法错（符号表溢出或标识符无定义）为止。9.链接技术就是在造表时，把冲突的各标识符连接到一条“链”上，查找时，沿着这条“链”查找。这种技术通常把表分为两个区，一部分称为链根区，存放杂凑函数值不同的那些标识符；另一部分称为链表区，

第十一章

1.控制流分析：为了减少冗余的计算，编译程序需要在编译阶段对生成的中间代码的可能的运行性状态进行分析，这一过程称为控制流分析。

2.数据流分析：与此同时编译程序还应分析伴随控制流的各种数据变化情况，这一过程称为数据流分析。

3.控制流图：为了研究指令在运行时所有组合的可能，将每条指令作为有向图的结点，指令 I 到 J 有一条有想边，当且仅当指令 J 在运行时能紧随指令 I 之后。由此得到的有向图称为控制流图。

4.编译程序的优化过程：编译程序的优化过程就是“分析→变换→再分析→再变换”的，如此往复。

5.**基本块**：没有任何其他的边引入其中间结点，也没有任何边从中间结点引出，其执行的程序不会被任何控制流所破坏。我们把满足上述条件的最大子图称为基本块。

6.**全局优化**：通过分析基本块之间的关系而进行的优化称为全局优化。

7.**局部优化**：全局优化对编译的过程整体进行优化，而在基本块内部进行的优化称为局部优化。

8.**公共子表达式**：设 x op y 为中间代码中赋值号右边的表达式，如果该表达式在中间代码链接中有两次出现，而且第一次出现到第二次出现的任意控制流中都没有对 x 和 y 进行直接或间接的赋值，则称 x op y 为公共子表达式。

9.活跃的、不活跃的：称某一变量 x 在三地址码某处 p 活跃的，当且仅当在控制流图中存在一条从 p 到出口的有向路径，该有向路径上至少有一条语句引用了 x 且从 p 到引用 x 间没有对 x 的定义语句，否则 x 是不活跃的。

10.**循环不变量**：再循环内的一条指令如果其计算结果与循环迭代的次数无关，则称为循环不变量。

11.**内循环**：前一个循环嵌套后一个循环，后者称为前者的内循环。

12.**代码外提**：在迭代程序中循环由不变量和常变量组成的表达式进行反复计算无疑是多余的，将它们外提到循环的入口之前进行一次计算即可。由此减少了循环内代码执行的次数。这样的优化方法称为代码外提。

13. **循环展开**：循环展开通过减少循环的迭代次数并增大循环体的代码，以减少对归纳变量的赋值和循环边界条件的检测而调高循环整体的执行效率。

第十二章

1. **四元式序列**：运算是按其执行顺序排列的。
2. **全局量 ACC**：它是编译时刻的变量，ACC 为空则表明运行时累加器为空；ACC 非空则表明它包含了当前累加器中的变量名或临时变量的值。
3. **过程 INACC**：其功能是在生成可交换的双边运算指令之前，调用该过程，它生成将其中任一运算量存入累加器的指令。
4. 在生成三元式 i 的代码之前，必须检查两个运算量，如果其中有一个运算量引用了前面的三元式，那么就必须分配给该三元式存放结果值的临时变量名去代替这个运算对象。
5. 生成三元式 i 的代码之后，还必须生产一个临时变量，用以描述执行该代码后所得到的结果值。
6. 树形表示生成代码：把单个表达式的 i 个三元组序列想象成一棵树，TR(i)表示跟分支，树形表示生成代码，用到一个动作表和一个递归过程，对树根为 TR(i)的子树生成代码。
7. **逆波兰**：从左至右顺次扫描逆波兰表示中的运算符和运算量，当扫描到运算量时，将运算量的名字存入运算量栈中，当扫描到一个二目运算时，利用栈项和两个运算量的名字生成相应的代码。
8. 任何算术运算或逻辑运算，其代码生成的一般过程可以归纳如下：
   ① 生成建立运算量地址的代码；
   ② 生成实现任何必须要的类型的转换代码；
   ③ 生成把一个运算量取到累加器中的代码；
   ④ 生成该运算的代码
9. 寄存器的分配：在计算一个表达式时，如何使用使用所需要的寄存器个数最少。
10. 算表达式如何使用所需存取令最少：先找出运算序列中涉及寄存器的那些指令，把运算序列变换成以这些指令为结点的一个有向图，所述问题就变成从图中找出某个结点到另一个结点的最短通路问题。

第十三章

1.字符串集上的一个分类称为单词，如标识符和整形常数等。单词的成员称为**词形**.

2.单词的描述称为**模式**。如标识符是由字母开头并有字母和数字组成的字符串。

3.整形常数是由数字组成的字符串，模式一般用正则表达式进行精确描述。

4.LEX 的输入文件称为 **LEX 源文件**

5.C 语言源程序输出到列名为 lex.yy.c 的文件中，该文件称为 LEX 的**输出文件**或输出的词法分析程序。

6.空格、横向跳格和换行符称为**白字符**。

7.**规则部分** 是 LEX 源文件的核心，它包括一组模式和在生成分析程序识别相应模式进行处理的 C 语言动作。

8.如果有多个模式可以匹配，则 yylex()将选择能匹配最长输入串的模式，称为**最长匹配原则**。

9.如果还有多个模式匹配长度相同的输入串，则 yylex()函数选择在 LEX 源文件中排列最前的模式进行匹配，称为**最先匹配原则**。

9.如果没有任何模式匹配输入的字符，则 yylex()使用 **缺省规则** 。

10.LEX 提供控制某些模式在一定状态下使用的功能，称为**条件模式**。

第十四章

1. 语法分析是对输入文件的二次重组，输入文件是**有序的字符串**。
2. YACC 的输入文件称为 YACC 源文件，它包含一组以 Backus-Naur 范式（BNF）书写的形式文法规则，以及对每条规则进行语义处理的 C 语言语句。
3. YACC 的输入文件有两个：一个是包含有语法分析函数 int yyparse()的 C 语言源程序 y.tab.c，称为**输出的语法分析程**序；另一个是包含有源文件中所有的终结符编码的宏定义文件 y.tab.h，称为**输出的单词宏定义头文件**。
4. 逆波兰表示的形式文法规则：input 由多行组成，每行或者有一个逆波兰表示或者为空行，每个逆波兰表示的语义值是该表达式计算后的结果，而在分析程序规约一个有表达式的行时，执行打印该表达式计算结果的语义动作。
5. 在 YACC 源文件中，有两种方式表示的单词：一种是在定义部分通过 YACC 指令%token 定义文法中出现的单词，称为**有名单词**；另一种是单个字符，称为**字符单词**。
6. 单词的**编码原则**是：字符单词使用其对应的 ASCII 码；有名单词的编码从 257 开始，并用 C 语言宏定义的方式实现编码，其宏名就是单词本身。根据 C 的宏定义的习惯，单词名一般用大写字母表示。为了区别单词，非终结符一般用小写字母组成的字符串表示。用户在对单词命名时还要注意，单词名一定不要和使用该单词名的 C 源程序中已有的宏名相同，否则在编译该 C 模块时会产生宏定义冲突。
7. YACC 产生的语法分析程序除了有一个保存移近单词和归约非终结符的分析栈之外，同时还有一个与分析栈并行的栈称为**语义栈**。用于存放与分析栈文法符号对应的语义值。
8. 设定 YYSTYPE 的宏定义的方式：（1）如果语义值的数据类型是简单类型，用户可直接在 C 语言代码部分用宏定义 YYSTYPE 为所需的数据类型。（2）如果不同的的语法符号有不同语义值，这些语义统称为**多类型语义值**。YACC 利用 C 语言的共同体结构实现对不同语法符号的语义值，选择不同的数据类型，这也是共同体结构一个最典型的应用。
9. YACC 提供在源文件中定义运算符号优先级别和结合次序的机制，来改变缺省的移进策略。
10. 语义动作，是指计算非终结符的语义值一般随归约动作同时进行，YACC 提供在每个语法规则的尾部附加 C 语言代码的功能。
11. 规则中部的语义动作，是指语义动作一般出现在规则的尾部，在分析程序归约某一句子成分时执行，YACC 也允许在生产式语法符号串的中部嵌入语义动作。
12. 含有 error 的规则称为**出错规则**。
13. 词法分析程序处理出错规则的方式：语法分析程序在分析 non-terminal 的语句成分出错时，如果这时栈顶的语法符号在 S_i 之前，则分析程序将保留当前输入并从分析栈中弹出相应的栈顶元素，直到当前单词和栈顶语法符号在分析表中有有效的动作为止；如果在

弹出栈顶元素时，分析栈已空，则分析程序将退出执行；如果这时栈顶元素是 S_i，则分析程序将把单词 error 压入栈中，用当前单词继续分析，这时分析程序将使用出错规则去匹配以后的语句；如果发生错误，栈顶的语法规则符号在 error 之后，在分析程序将试图放弃当前输入，读取下一个单词，知道出现一个有效的单词为止。