作者：希赛教育软考学院　来源：希赛网　2014年05月20日

考点分析

第5章 程序设计语言

根据考试大纲，在程序设计语言方面，主要考查以下内容：

（1）汇编、编译、解释系统的基础知识和基本工作原理。

（2）程序设计语言的基本成分：数据、运算、控制和传输，程序调用的实现机制。

（3）各类程序设计语言的主要特点和适用情况。

5.1 考点分析

本节把历次考试中程序设计语言方面的试题进行汇总，得出本章的考点，如表5-1所示。

表5-1 程序设计语言试题知识点分布

考试时间	分数	考查知识点
10.11	2	非确定性有限自动机（2）
11.05	3	确定性有限自动机（2）、正规式（1）
11.11	3	词法分析（1）、有限自动机（2）
12.05	4	逻辑式语言（1）、语法与语用（2）、文法（1）
12.11	3	语言的分类（1）、自动机和正规式（2）
13.05	3	语言比较（1）、函数调用（1）、循环语句（1）
13.11	9	C语言结构体（1）、C和C++的比较（2）、动态语义（1）、语言的分类（1）、正规式（2）、全局变量（1）、上下文无关文法（1）
14.05	5	编译器（1）、文法（1）、程序语言的分类（1）、有限自动机（1）、确定性有限自动机（1）

根据表5-1,我们可以得出程序设计语言的考点主要有以下5个：

（1）基本概念：包括汇编、编译、解释系统的基础知识和基本工作原理。

（2）语言的分类：包括各种语言的特点和应用场合。

（3）控制结构：包括语句结构、函数调用、全局变量等。

（4）文法：包括文法的类型，主要考查上下文无关文法。

（5）自动机与正规式：包括非确定性有限自动机、确定性有限自动机、正规式。

对这些知识点进行归类，然后按照重要程度进行排列，如表5-2所示。其中的星号（*）代表知识点的重要程度，星号越多，表示越重要。

表5-2 程序设计语言各知识点重要程度

知识点	10.11	11.05	11.11	12.05	12.22	13.05	13.11	14.05	合计	比例	重要程度
基本概念			1	2				1	4	12.50%	★★
语言的分类				1	1	1	4	1	8	25.00%	★★★
控制结构						2	2		4	12.50%	★★
文法				1			1	1	3	9.38%	★
自动机与正规式	2	3	2		2		2	2	13	40.63%	★★★★★

在本章的后续内容中，我们将对这些知识点进行逐个讲解。但有关C语言和C++语言的语法知识，不在这里讲解。请读者学习希赛教育的《C语言程序设计辅导视频教程》和《面向对象程序设计（C++版）辅导视频教程》。

第 5 章：程序设计语言 作者：希赛教育软考学院 来源：希赛网 2014年05月20日

基本概念

5.2 基本概念

程序设计语言用以书写计算机程序（指计算任务的处理对象和处理规则的描述），它包括语法、语义、语用三个方面。语法表示程序的结构或形式，即表示构成语言的各记号间的组合规则，但不涉及这些记号的特定含义，也不涉及使用者。语义表示程序的含义，即表示按照各种方法所表示的各个记号的特定含义，但不涉及使用者。语用表示程序与使用者的关系。

程序设计语言的基本成分有数据、运算、控制和传输。数据成分用以描述程序中所涉及的数据；运算成分用以描述程序中所包含的运算；控制成分用以表达程序中的控制构造；传输成分用以表达程序中数据的传输。

第 5 章：程序设计语言 作者：希赛教育软考学院 来源：希赛网 2014年05月20日

编译系统基础知识

5.2.1 编译系统基础知识

编译程序的职能是把使用某程序设计语言书写的程序翻译为等价的机器语言程序，所谓等价是指目标程序执行源程序的预定任何。一般来说，编译程序分为以下几个部分：词法分析、语法分析和语义分析、代码优化、代码生成和符号表管理。各部分之间的关系如图5-1所示。

词法分析程序是编译程序的第一个部分，它的输入是源程序中由字符组成的符号。编译程序需要把程序的这种外部形式转换成适合后续程序处理的形式，其功能有：

（1）识别出源程序中意义独立的最小词法单位--单词，并且确定其类型（例如是标识符、关键字、操作符还是数字等）。

（2）删除无用的空格、回车和其他与输入介质有关的无用符号，以及程序注释。

（3）报告分析时的错误。

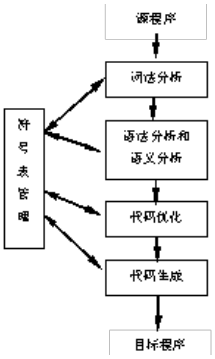

图5-1 编译系统

经过词法分析程序处理后，源程序就转化为单词串。每个单词都是一个意义独立的单位，其所

包含的信息量个数固定。语法分析程序根据特定程序设计语言的文法规则，检查单词串是否符合这些规则。一旦语法分析程序分解出其中一个文法结构，该结果的语义分析程序就进行相应的语义检查，在有需要的时候输出相应的中间代码。这里的中间代码可以理解为假想的虚拟机的指令，其执行次序反映了源程序的原始定义。语法和语义分析程序是编译程序中的关键部分。

中间代码作为代码生成程序的输入，由代码生成程序生成特定的计算机系统下的机器代码。为了提高目标代码的运行效率和减小目标代码大小，也可以在语法语义分析程序与代码生成程序之间插入代码优化程序。代码优化程序在不改变代码所完成的工作的前提下对中间代码进行改动，使其变成一种更有效的形式。

编译程序在完成其任务的过程中，还需要进行符号表的管理和出错处理。在符号表中登记了源程序中出现的每一个标识符及其属性。在整个编译过程中，各部分程序都可以访问某标识符的属性，包括标识符被说明的类型、数组维数、所需存储单元数，所分配的内存单元地址等。错误管理程序是在分析程序发现源程序有错误而无法继续工作时进行其工作的。其任务是记录并向用户报告错误及其类型和位置，或者尝试进行某种恢复工作。

解释系统基础知识

5.2.2 解释系统基础知识

解释程序是一种语言处理程序，它实际是一台虚拟的机器，直接理解执行源程序或源程序的内部形式（中间代码）。因此，解释程序并不产生目标程序，这是它和编译程序的主要区别。图5-2显示了解释程序实现的3种可能情况。

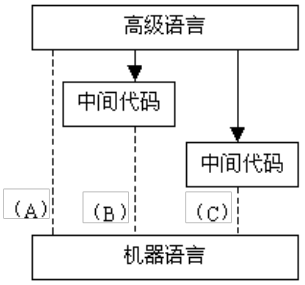

图5-2 解释系统

在类型A的实现方案中，解释程序直接对源程序进行解释执行。这种解释程序对源程序进行逐字符的检查，然后执行程序指令表示的动作。例如，当解释程序扫描到字符串序列：

GOTO Label

解释程序意识到GOTO命令代表无条件跳转至Label所标识的位置，于是就开始搜索源程序中标号Label后面紧跟冒号":"的出现位置，然后跳转至该位置继续执行。这类系统在实现时需要反复扫描源程序，因此按类型A方案实现的解释程序效率很低，早期的解释性Basic语言就是采取该方案实现的。

在类型B的实现方案中，翻译程序先将源程序翻译成较贴近高级语言的高级中间代码，然后再扫

描高级中间代码，对高级中间代码进行解释执行。所谓较贴近高级语言的高级中间代码，是指中间代码与高级语言的语句形式相像，两者存在着一一对应的关系。

类型C又是一种解释程序的实现方案。类型C的解释程序和类型B的解释程序不同点在于：类型C的解释程序首先将源程序转化成和机器代码十分接近的低级中间代码，然后再解释执行这种低级中间代码。一般说来，在这种实现方案下，高级语言的语句和低级中间代码之间存在着1:n对应关系。例如微软的C#语言，首先被编译成一种形式上较类似汇编语言的中间语言IL表示的代码，然后通过通用语言运行时（Common Language Runtime,CLR）解释执行IL程序。这类系统具有良好的可移植性。

作者：希赛教育软考学院　来源：希赛网　2014年05月20日

语言的分类

5.3 语言的分类

可以从不同的角度对程序设计语言进行分类，从程序语言的本质上来看，可以分为三类：机器语言、汇编语言和高级语言。

机器语言是特定计算机系统所固有的语言，用机器语言编写的程序可读性很差，程序员难以修改和维护。

汇编语言用助记符号来表示机器指令中操作码和操作数，汇编语言仍然是一种和计算机的机器语言十分接近的语言，它的书写格式在很大程度上取决于特定计算机的机器指令。

目前已有许多流行的高级语言，如Fortran、Cobol、Pascal、C/C++、Java等。这类语言与人们的自然语言比较接近，大大提高了程序设计的效率，便于进行交流。Fortran是第一个被广泛用于科学计算的高级语言。Algol 60是另一个早期研制出来的高级语言。它有严格的文法规则，用巴科斯范式BNF来描述语言的文法。Algol 60是一个分程序结构的语言。COBOL是一种面向事务处理的高级语言。Pascal语言提供的为数不多而又相当紧凑的机制使得这个语言具有相当强的表达能力。C是一种通用程序设计语言。C作为一种较低级的语言，提供了指针和地址操作的功能。C提供书写结构良好的程序所需的控制结构。C与Unix操作系统紧密相关，Unix操作系统及其上的许多软件都是C编写的。

1.过程性语言

过程性语言就是指传统的程序设计语言。在使用传统程序设计语言时，程序员不仅要说明信息结构，而且要描述程序的控制流程。因此它也被称为过程性语言。过程性语言是相对于新型程序设计语言（函数式、逻辑式、面向对象）和第四代语言（4GL）而言的，其特点是通过使用赋值语句改变变量的状态，来完成各种任务。

2.面向对象语言

Simula是最早提出类的概念的语言，完备地体现面向对象并提出继承概念的程序设计语言是Smalltalk 80,C++和Java是目前用得最多的面向对象的语言。有关面向对象的特性，我们将在第9章进行讨论。

3.逻辑型语言

逻辑型语言是一类以形式逻辑为基础的语言，其理论基础是一阶谓词演算。Prolog是典型的逻辑式语言，具有和传统的命令型程序设计完全不同的风格。组成Prolog程序的语句的基本形式是Horn子句，Prolog程序由围绕某一主题的事实、规则和询问三类语句组成，这三类语句分别用来陈述事实、定义规则和提出问题。Prolog具有很强的推理功能，适用于书写自动定理证明、专家系统、自然语言理解等人工智能问题的程序。

归约的方法是逻辑式的语言中主要方法之一。它是把一簇命题转换成标准的子句集形式，采用匹配和合一的算法，消除冗余，以获得新命题正确性的证明或命题集一致性的验证。

4.函数型语言

函数型程序语言是一类以λ演算为基础的语言。LISP是典型的函数型程序语言。函数是一种对应规则（映射），它使其定义域中每一个值和值域中唯一的值相对应。

函数型程序设计语言的优点之一是它是一种面向值的语言，无状态，无副作用，具有引用透明性，函数值只取决于变元值，具有相同一组变元的函数，基值唯一。对表达式中出现的任何函数都可以用其他函数来代替，只要这些函数调用产生相同的值。这些特点有助于程序模块化的实现。

函数式程序设计语言和逻辑式程序设计语言都属于申述式语言。

5.可视化开发工具

目前，比较热门的软件开发工具都是可视化的，例如：Visual Basic、Visual C++、Delphi、Power Builder和JBuilder等。这些工具是一种事件驱动程序语言，编程时，在程序内必须设计各种事件的处理程序代码。当此事件发生时，随即驱动执行相应的程序段。这些开发工具都提供了良好的控件工具，用户可以很方便地建立用户界面，大大提高了程序设计的效率。

上一节　本书简介　下一节

控制结构

5.4 控制结构

在控制结构方面，主要考查三种基本结构、变量与常量，以及函数调用的方式。

上一节　本书简介　下一节

常量与变量

5.4.1 常量与变量

常量是在程序运行过程中值不修改的数据。常量具有名字，便于望文生义，也可以方便的引

用，需要时仅修改常量定义处，因此可以提高程序的可读性以及可维护性。

变量是在程序运行过程中可以修改的数据。变量也具有名字，变量可以分为局部变量和全局变量。

局部变量也称为内部变量。局部变量是在函数内作定义说明的。其作用域仅限于函数内，离开该函数后再使用这种变量是非法的。

全局变量也称为外部变量，它是在函数外部定义的变量。它不属于哪一个函数，它属于一个源程序文件。其作用域是整个源程序。在函数中使用全局变量，一般应作全局变量说明。只有在函数内经过说明的全局变量才能使用。

在程序中，函数中定义的变量是局部变量，在函数之外定义的变量为外部变量，外部变量是全局变量。二者的有效范围不一样，前者只在本函数范围内有效，后者的有效范围从定义该全局变量的位置开始到本源文件结束。

一般变量处于内存中，如果某一变量被频繁使用，则可以将其定义为寄存器变量，将变量放在寄存器中，由于运算器访问寄存器的速度远远大于访问内存的速度，可以节省不少时间。

第 5 章：程序设计语言　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

三种控制结构

5.4.2 三种控制结构

表达式的顺序控制是把操作数和操作符看做基本单位，研究操作符的计算顺序。而语句间的顺序控制是把程序语句作为基本单位，研究语句执行的顺序。程序语言对语句的执行都遵循一条隐含规则，这就是在没有其他顺序控制结构规定的情况下，按照语句在程序中的物理位置执行程序，也就是顺序执行。改变这种语句执行次序的方法是使用程序顺序控制结构，这些控制结构有跳转结构、选择结构（分支结构）和循环结构（重复结构）。其中，跳转结构是不主张使用的。因此，程序的基本控制结构主要有顺序结构、选择结构和循环结构

1.跳转结构

跳转结构就是令程序控制无条件的从当前语句转向给定的语句执行的控制结构，跳转语句的执行非常有效，它反映了计算机本身硬件的转移指令，如x86指令中的jmp指令。通常的跳转语句都有如下形式：goto <标号>,Fortran和C语言等都提供了goto语句。当程序控制遇到goto语句时，会转移到标号所指出的相应语句继续执行。

虽然goto语句的使用十分简单和高效，但是大量的使用会令程序控制逻辑混乱，程序变得难以理解和维护。人们已经证明可以使用顺序结构、选择结构和循环结构组成任何程序，而抛弃掉"有害的"goto语句。目前比较一致的观点是程序员必须谨慎地使用goto语句，使用时必须考虑是否可以用更好的结构来代替。

2.选择结构

选择结构是对给定条件进行判断，然后根据结果执行不同的语句或语句块的结构。最典型的选择结构的形式如下：

If <expr> then

<statements1>

Else

<staements2>

Endif

意味着如果expr条件为真则执行statements1语句块，否则执行statements2语句块。在某些复杂的情况下，需要对多个条件进行判断，则if-then-else语句会进一步复杂，演化为if-then-elseif-then-else等。

在两个分支的选择结构基础上，多数语言也会提供多分支的选择结构，它在许多情况下可以改善程序的可读性。典型的多分支选择结构如下：

Switch（<expr>）

Case of result1:

<statements1>

Case of result2:

<statements2>

…

Default:

<default_statements>

Endswitch

虽然case控制结构的功能可以由if-then-else结构来模拟，但是case控制结构能提供更清晰的计算过程的反映。

C/C++的情况比较特别，在case结构中使用break语句表示跳出结构的控制，如果在其中一个case中没有使用break语句，则控制会顺序执行至下一个case中的语句。这个特点在为程序员带来方便的同时，也为程序员带来了麻烦。程序员疏忽漏掉的break语句会导致程序有意想不到的执行结构。因此在C#中，语言再不允许这种case的"贯穿",而强制程序员使用goto语句跳转至相应的case标号，以保证程序员清楚知道程序控制的行为。

3.循环结构

循环结构是根据条件重复执行指定语句的控制结构。循环结构是由循环头和循环体组成的。循环头就是循环的条件，用以控制循环的次数，循环体则是提供动作的语句。典型的循环头结构有以下几种：

（1）计数器循环

这种结构需要说明一个循环计数器，并且在头部中说明计数器的初值、终值和增量。典型的计数器循环的结构是Pascal的计数器循环：

for i:=0 to 30 step 2 Do <body>

该循环的头部说明了计数器为i,其初值为0,终值为30,增量为2,循环的执行次数为16次。

（2）条件循环

条件循环是指在给出的条件表达式成立时重复执行循环体的循环结构，它的头部中说明了该条件表达式。这种循环期望在循环体执行的时候会改变条件测试表达式中的某个变量的值，否则循环将永不终止。典型的条件循环的结构有两种：一是While <expr> do <body>,另一种是Repeat

<body> until <expr>.前者是先测试条件，然后执行循环体，循环体执行0次或0次以上。而后者是先执行循环体，再测试条件，循环体执行1次或1次以上。

（3）基于数据的循环

基于数据的循环的循环次数是由数据格式决定的。例如C#中的foreach结构：

foreach（object o in <collection>）{...}

对于每一次循环，变量o都会取得数据集中的下一个值，数据集元素的个数决定了循环的次数。

（4）不定循环

如果循环结束条件过于复杂，不容易在头部表示，通常会使用在循环头部没有显示终止测试的无限循环，然后在循环体中通过条件判断退出循环。C/C++中有两种典型的不定循环结构：一是for（；；）<statements>,另外是while（1）<statements>.

函数调用

5.4.3 函数调用

在程序设计中，习惯把程序看做是层次结构，程序从主程序开始执行，然后进入各层次的过程执行，到最后返回主程序结束。过程为程序员提供了一种抽象手段，其实际上是一组输入到一组输出的映射。

过程通常有4个要素：过程名、过程体、形式参数列表和返回值类型。例如C语言中的函数（即C语言中的过程）如下：

int Function1（int x, int y）；

其中Function1为函数名，（int x, int y）为形式参数列表，int是返回值类型。

当用户调用一个过程时，就会发生通过参数传递信息的过程之间的通信。形式参数就是过程定义中用于命名所传递的数据或其他信息的标识符，而实际参数是在调用点表示向被调用过程传递的数据或其他信息的表达式。在大多数的语言中，形式参数和实际参数之间的对应关系通常按位置来确定。程序语言传递参数的方式通常有传值调用、引用调用和传值-结果调用。

1.传值调用

在按值调用（call by value）时，过程的形式参数取得的是实际参数的值。在这种情况下，形式参数实际上是过程中的局部量，其值的改变不会导致调用点所传送的实际参数的值发生改变。也就是数据的传送是单向的。在C语言中只有按值调用的过程参数传递方式。

2.引用调用

在按引用调用（call by reference）时，过程的形式参数取得的是实际参数所在单元的地址。在过程中，对该形式参数的引用相当于对实际参数所在的存储单元的地址引用。任何改变形式参数值的操作都会反映在该存储单元中，也就是反映在实际参数中，因此数据的传送是双向的。C++语言既支持按值调用，也支持按引用调用。

3.传值–结果调用

传值-结果调用也成为拷入/拷出，因为初始时实际参数被拷贝到形式参数，而在过程调用结束时再把形式参数拷贝回实际参数。

例如：函数t（）、f（）的定义如图5-3所示，若调用函数t（）时传递给x的值为3,并且调用函数f（）时，第一个参数采用传值方式，第二个参数采用传引用方式，则函数t的返回值为24.分析如下：

开始调用t（）函数时，x=3,a=8,调用f（）的两参数值分别是f（3,8）；但要注意：f（）函数的第二个参数是采用传引用方式，所以要看成f（）函数的s和t（）函数的a共用一段内存地址。

f（）函数执行之后，s=21,即a=21,所以最后t（）函数的返回值24.

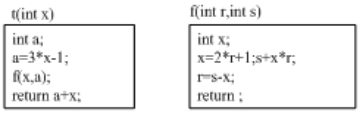

图5-3 函数调用的例子

上一节　本书简介　下一节

文法

5.5 文法

如何来描述一种语言？如果语言是有穷的（只含有有穷多个句子），可以将句子逐一列出来表示；如果语言是无穷的，找出语言的有穷表示。语言的有穷表示有两个途径：一是生成方式（文法），语言中的每个句子可以用严格定义的规则来构造。二是识别方式（自动机），用一个过程，当输入任意字符串属于语言时，该过程经有限次计算后就会停止并回答"是".若不属于语言时，则要么能停止并回答"不是",或永远不停地继续下去，这种方式将在5.6节中讨论。

上一节　本书简介　下一节

文法的定义

5.5.1 文法的定义

一个形式文法G是一个四元组G=（N,T,P,S），其中

（1）N是一个非空的有穷的非终结符集合，它的元素称为变元或非终结符。

（2）T是一个非空的有穷的终结符集合，且N∩T=Φ，T的元素称为终结符。

（3）P是一个非空的有穷的产生式集合，它的元素称为产生式或改写规则，形式为

α→β

这里α，β∈（N∪T）*且α≠ε。

（4）S是起始符且S∈N.有时，将文法简记为G（S）。

形式文法是语言的产生系统，用以产生出符合要求的句子。

1.句型与短语

给定文法G=（N,T,P,S），设ω、λ∈（N∪T）*,若存在j、x∈（N∪T）*和P中的产生式α→β，使得ω=jαx,λ=jβx,即把ω中的α改写成β后得到λ，则称λ是ω直接推导出来的，记作ω？λ。

设G=（N,T,P,S）是文法，$\alpha_1,\alpha_2,\ldots$，αn都是（N∪T）*中的字符串，n≥1且$\alpha1 \overset{\cdot}{\Rightarrow} \alpha2 \overset{\cdot}{\Rightarrow} \ldots \overset{\cdot}{\Rightarrow}$ αn,则称αn是由α1推导出来的，记作$\alpha_1 \overset{\cdot}{\Rightarrow} \alpha_n$,其中$\alpha_i$直接推导出$\alpha_{i+1}$（1≤i≤n+1）。注意：对（N∪T）*中的所有字符串ω，都有$\omega \overset{\cdot}{\Rightarrow} \omega$。$\overset{\cdot}{\Rightarrow}$是$\overset{\cdot}{\Rightarrow}$的自反传递闭包。设文法G=（N,T,P,S），若S $\overset{\cdot}{\Rightarrow}$x,则称x是文法G的句型。

如果在推导的任何一步$\alpha \overset{\cdot}{\Rightarrow} \beta$，其中α、β是句型，都是对α中的最左非终结符进行替换，则称为最左推导。同样的，如果在推导的任何一步$\alpha \overset{\cdot}{\Rightarrow} \beta$，其中α、β是句型，都是对α中的最右非终结符进行替换，则称为最右推导。

从一个句型到另一个句型的推导过程并不是唯一的，为了对句子的结构进行确定性分析，我们往往只考虑最左推导或最右推导，并且称最右推导为规范推导。由规范推导所得的句型称为规范句型。

如果SαAδ且Aβ，则称β是句型$\alpha \overset{\cdot}{\Rightarrow} \beta A\delta$相对于非终结符A的短语或简称β是句型$\alpha \overset{\cdot}{\Rightarrow} \beta A\delta$的一个短语。特别情况，当有A→β产生式时，β为句型$\alpha \overset{\cdot}{\Rightarrow} \beta A\delta$的一个直接短语或简单短语。如果短语中含有终结符，而且它又不存在具有同样性质的真子串，则称该短语为素短语。注意，短语的两个条件缺一不可。仅有$A \overset{\cdot}{\Rightarrow} \beta$未必意味着β就是句型$\alpha \overset{\cdot}{\Rightarrow} \beta A\delta$的一个短语，还需要有SαAδ这一条件加以限制，也即短语属于该句型的组成部分。

一个句型的最左直接短语称为该句型的句柄。如果对$S\alpha A\delta \overset{\cdot}{\Rightarrow} \alpha \overset{\cdot}{\Rightarrow} \beta\delta$来说，将句型$\alpha \overset{\cdot}{\Rightarrow} \beta A\delta$中的句柄β用产生式的左部符号代替便得到新句型αAδ，这是一次规范归约，恰好是规范推导的逆过程。注意，一个句型的直接短语可能不只一个，但最左直接短语则是唯一的。

若Sx,且x∈，则称x是文法G的句子。

例如：对文法G=（{E},{i,（，）},P,E），其中P的产生式如下：

E→E+E|E*E|（E）|i,则句子i+i*i按文法的最左推导是：

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

句子 $i+i*i$ 按文法的最右推导是：

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

在句型E+E*i中，i、E*i和E+E*i是句型E+E*i的三个短语；其中，i为素短语；E*i虽为短语且含有终结符，但它的真子串i是素短语，故E*i不是素短语；同样E+E*i也不是素短语。开始符S本身只能是文法的一个句型而不可能是一个句子；此外，上面推导出的i+i*i是文法G的一个句子（当然也是一个句型），而E+E、E+E*E、E+E*i和E+i*i都是文法G的句型。

2.语法树

语法树以图示化的形式把句子分解成各个组成部分来描述或分析句子的语法结构，这种图示化的表示与所定义的文法规则完全一致，但更为直观和完整。

对文法G=（N,T,P,S），满足下列条件的树称为G的语法树：

（1）每个节点用G的一个终结符或非终结符标记。

（2）根节点用文法开始符S标记。

（3）内部节点（指非树叶节点）一定是非终结符，如果某内部节点A有n个分支，它的所有子节点从左至右依次标记为$x_1,x_2,\ldots，x_n$,则$A\rightarrow x_1x_2\ldots x_n$一定是文法G的一条产生式。

（4）如果某节点标记为ε，则它必为叶节点且是其父节点的唯一子节点。

对应于一个句型的语法树是以文法的开始符S作为根节点的，并随着推导逐步展开；当某个非终结符被它对应的产生式右部的某个候选式所替换时，这个非终结符所对应的节点就产生出下一代新节点，即候选式中从左至右的每一个符号都依次顺序对应一个新节点，且每个新节点与其父节点之间都有一条连线（树枝）。在一棵语法树生长过程中的任何时刻，所有那些没有后代的树叶节点自左至右排列起来的就是一个句型。

例如：上述例子中的句子$i+i^*i$相应的语法树如图5-4所示。

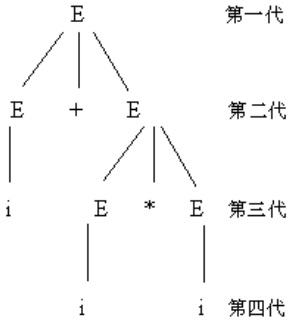

图5-4 句子$i+i^*i$的语法树

在构造语法树时可以发现，一个句型的最左推导及最右推导只是决定先生长左子树还是先生长右子树，句型推导结束时相应的语法树也随之完成，这时已不能看出是先生长左子树还是先生长右子树，所呈现的只是已经长成的这个句子或句型的语法树。这与使用文法规则进行推导是有差异的，使用文法规则的推导过程是有先后之分的。下面再看一个例题。

假设某程序语言的文法G=（N,T,P,S），其中：N={a,b,d,（，）},T={S,L},S是开始符号。产生式集合P定义如下：

S→a|b|（L）

L→LdS|S

句型（Sd（L）db）的语法树如图5-5所示。

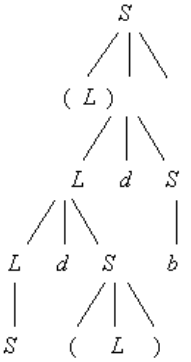

图5-5 句型（Sd（L）db）语法树

这个句型是由S经一般推导推出来的，其步骤如下：

S→（L）→（LdS）→（Ldb）→（LdSdb）→（Ld（L）db）→（Sd（L）db）

句型（Sd（L）db）不是从S的最左推导，也不是从S的最右推导，因为句型（Sd（L）db）的

第一步肯定是由S→（L）→（LdS）得出，按照最左推导的规则（Lds）→（LdSdS）→（SdSdS），最终不可能推出（Sd（L）db）的句型；按照最右推导的规则（Lds）→（Ldb）→（Ld（L）db），最终也不可能推出（Sd（L）db）的句型。

从语法树看见，S,（L），b,Sd（L），Sd（L）db,（Sd（L）db）都是短语。但无论如何，无法从S推导出d（L）、Ld或Sd（L）d,所以它们不是短语。其中S是句型相对于规则L→S的直接短语，也是最左直接短语，所以S是句柄。（L）是句型相对于规则S→（L）的直接短语，b是句型相对于规则S→b的直接短语。（L），b是素短语。

文法的分类

5.5.2 文法的分类

设G=（N,T,P,S）是文法，称

L（G）={ω|ω∈T^*且S$\overset{+}{\Rightarrow}$ω}

是由文法G产生的语言，记为L（G）。L（G）中的每一个字符串，必是由终结符组成并且是从起始符S推导出来的句子，即语言是句子的集合。如果L（G1）=L（G2），则称文法L（G1）和L（G2）等价。

对形式文法进行分类的最常见的是诺姆·乔姆斯基于1956年发展的乔姆斯基谱系，这个分类谱系把所有的文法分成四种类型：无限制文法也称短语结构文法、上下文相关文法、上下文无关文法和正则文法。这四种文法类型的区别在于对产生式附加了不同的限制，它们依次拥有越来越严格的产生式规则，同时文法所能表达的语言也越来越少。四类文法中最重要的是上下文无关文法和正则文法。

0型文法也称为无限制文法或短语结构文法，0型文法产生的语言称作0型语言，也称递归可枚举语言。0型文法是一般的形式文法，对文法定义的产生式规则不附加任何条件。

1型文法，也称上下文相关文法，如果文法G中的每一个产生式α→β，有|α|≤|β|则称G是一个1型文法。如果L-｛ε｝可由1型文法产生，则L称作1型语言或上下文相关语言。这种文法的产生式规则形如

αAβ→αγβ

这里的A∈N是非终结符号，α，β，γ∈（N∪T）*且γ≠ε是包含非终结符号与终结符号的字符串；α、β可以是空串，但γ必须不能是空串。也就是在这种文法中，在替换变元时必须考虑这个变元的上下文。产生式αAβ→αγβ表示只有当A的上下文分别是α和β时，才能将A替换γ。

2型文法也称为上下文无关文法，如果文法G中的每一个产生式都形如

A→α

其中A∈N,且α∈（N∪T）*,则称G是一个2型文法，2型文法产生的语言称作2型语言或上下文无关语言。用产生式A→α把A替换成α时，不需要考虑A的上下文。

3型文法也称为正则文法，分为左线性文法和右线性文法。如果文法G中的每一个产生式都形

如：

A→αB或A→α

其中A,B∈N,且α∈$(T)^*$，则称G是一个右线性文法。如果文法G中的每一个产生式都形如

A→Bα或A→α

其中A,B∈N,且α∈$(T)^*$，则称G是一个左线性文法。

右线性文法和左线性文法都称为3型文法或正则文法，3型文法产生的语言称作3型语言或正则语言。

上下文无关文法和正则文法在程序语言的设计和编译理论中起着重要作用。词法分析可用正则文法解决，而语法分析使用上下文无关文法。

上一节　本书简介　下一节

自动机与正规式

5.6 自动机与正规式

对于本知识点主要是掌握正规式和有限自动机的基础知识与原理，掌握正规式与文法、自动机之间的转换，以及DFA和NFA之间的转换。

上一节　本书简介　下一节

有限自动机

5.6.1 有限自动机

有限状态自动机是具有离散输入和输出的系统的一种数学模型。系统可以处于内部状态的任何一个之中，系统当前状态概括了有关过去输入的信息，这些信息对在后来的输入上确定系统的行为是必需的。有限状态自动机与词法分析程序的设计有着密切的关系。下面是确定的有限状态自动机的形式定义：

一个确定的有限状态自动机（DFA）M是一个五元组：

M=（Σ，Q,q_0,,F,δ）

其中：

（1）Q是一个有限状态集合。

（2）Σ是一个字母表，其中的每个元素称为一个输入符号。

（3）q_0∈Q,称为初始状态。

（4）$F \subseteq Q$，称为终结状态集合。

（5）δ是一个从Q×Σ（Q与Σ的笛卡儿乘积）到Q的单值映射：

δ（q,a）= q（q,q'∈Q,a∈Σ）

表示当前状态为q,输入符号为a时，自动机将转换到下一个状态q',q'称为q的一个后继。

若$Q=\{q_1,q_2,\cdots,q_n\}$，$\Sigma=$ {a1,a2,...，an},则（δ（q_i,a_j））n×m是一个n行m列矩阵，称为M的状态转换矩阵，或称转换表。

有限状态自动机可以形象地用状态转换图表示，设有限状态自动机：

DFAM =（{S, A, B, C, f}, {1, 0}, S, {f}, δ），

其中δ（S,0）=B,δ（S,1）=A,δ（A,0）= f,δ（A,1）=C,δ（B,0）=C,δ（B,1）=f,δ（C,0）=f,δ（C,1）= f.

其对应的状态转换图如图5-6所示。

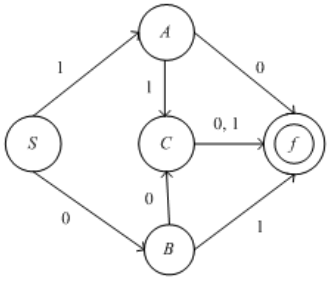

图5-6 状态转换图

图5-6中的圈表示状态节点，其中双圈表示终结状态节点。而边表示状态的转换，代表映射。边上的符号表示此转换需要输入的符号，代表映射的输入。

对于Σ上的任何字符串$w\in\Sigma^*$，若存在一条从初态节点到终态节点的路径，在这条路径上所有边的符号连接成的符号串恰好是w,则w被M所识别（或接受、读出）。M所能识别的符号串的全体记为L（M），称为M所识别的语言。如果对所有$w\in\Sigma^*$，以下述的递归方式扩张δ的定义：

δ（q,ε）=q

δ（q,wa）=δ（δ（q,w），a），对任何a∈Σ，q∈Q

我们则可以把DFA M所识别的语言形式定义为：

$$L(M)=\{w\mid w\in\Sigma^*，\text{若存在}q\in F，\text{使}\ \delta(q_0,w)=q$$

前面介绍的是确定的有限自动机，即一个状态对于特定的输入字符有一个确定的后继状态。而当一个状态对于特定的输入字符有一个以上的后继状态时，我们称该有限自动机为非确定有限自动机（NFA），其形式定义如下。

一个非确定的有限自动机M是一个五元组：

M=（Σ，Q,q_0,,F,δ）

其中Σ、Q、q_0、F的意义和DFA的定义一样，而δ是一个从$Q\times\Sigma$到Q的子集的映射，即δ：$Q\times\Sigma\to 2^Q$,其中2^Q是Q的幂集，即Q的所有子集组成的集合。

与确定的有限自动机一样，非确定有限自动机同样可以用状态转换图表示，所不同的是在图中一个状态节点可能有一条以上的边到达其他状态节点。同样，对于任何字符串$w\in\Sigma^*$，若存在一条从初态节点到终态节点的路径，在这条路径上的所有边的符号连接成的符号串恰好是w,则称w为M所识别（或接受或读出）。若q_0∈F,这时q_0既是初始状态，也是终结状态，因而有一条从初态节点到终态节点的ε–路径，此时空符号串可以被M接受。M所能识别的符号串的全体为L（M），称为M所识别的语言。

对任何一个M,都存在一个M'使L（M'）=L（M），这时我们称M'与M等价。构造与M等价的M'的基本方法是让M'的状态对应于M的状态集合，即如果有$\delta(q,a)=\{q_1,q_2,\cdots,q_n\}$，则$\{q_1,q_2,\cdots,q_n\}$把

有限自动机

上一节　本书简介　下一节

正规表达式

5.6.2 正规表达式

正规表达式是一个十分有用的概念，它紧凑地表达有限自动机所接受的……正规表达式的递归定义为一个正规表达式是按照一组定义规则由一些较简单的正规表达式……。在字母表Σ上的正规表达式可以使用以下规则定义。

（1）ε和Φ是Σ上的正规表达式，它们所表示的语言分别为{ε}和Φ。

（2）如果a是Σ内的一个符号，则a是一个正规表达式，所表示的语言为{a}，即包含符号串a的集合。

（3）如果r和s分别是表示语言L（r）和L（s）的正规表达式，那么：

（r）|（s）是一个表示L（r）∪L（s）的正规表达式。

（r）（s）是一个表示L（r）L（s）的正规表达式。

（r）*是一个表示（L（r））*的正规表达式。

（r）是一个表示L（r）的正规表达式。

通常在正规表达式中，一元运算符"*"具有最高的优先级，连接运算具有次优先级，运算符"|"具有最低优先级，这三个运算都是左结合的。每一个正规表达式R都对应一个有限自动机M,使M所接受的语言就是正规表达式的值。经过以下步骤可以从一个正规表达式R构造出相应的有限自动机M.

首先定义初始状态S和终止状态f,并且组成有向图，如图5-7所示。

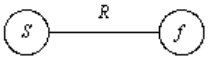

图5-7 有向图

然后反复应用以下规则，如图5-8所示。

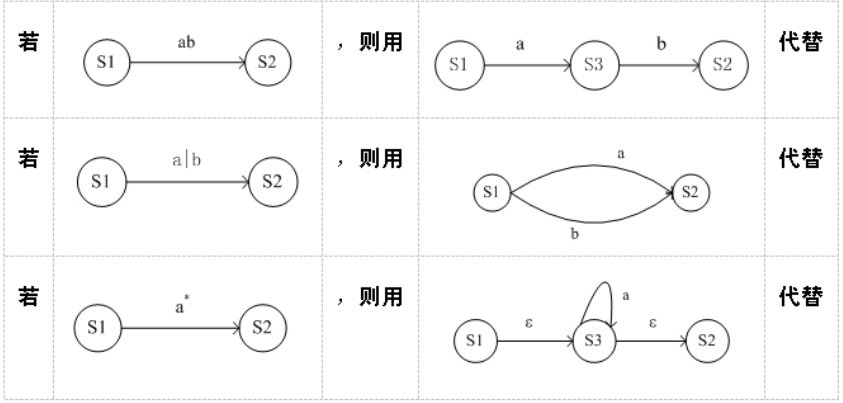

图5-8 正规表达式使用规则

直到所有的边都以Σ中的字母或ε标记为止。由此产生了一个带ε-转移的非确定有限自动机，然后可以通过上面介绍的方法，把该自动机转换成确定有限状态自动机。

下面举一个例子说明自动机理论在词法分析程序中的应用。C语言中对标识符的规定为由"_"或以字母开头的由"_"、字母和数字组成的字符串，该标识符的定义可以表示为下面的正则表达式：

$$(\_ \mid a)(\_ \mid a \mid d)^*$$

式中的α代表字母字符{A,…，Z,a,…，z},d代表数字字符{0,1,…，9}.利用前面的方法构造出如图5-9所示的有限自动机。

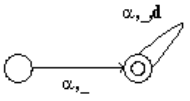

图5-9 有限自动机图例

该自动机所接受的语言就是C语言中的标识符。在有限自动机的状态转换过程中，需要执行相关的语义动作。例如当识别到一个标识符时，需要在符号表中添加该标识符，并且向语法分析程序输送表示该标识符的单词。

下面，我们再看一个例子。

假设某一确定有限自动机的状态转换图如图5-10所示，该DFA接受的字符串集是什么？与之等价的正规式是什么？

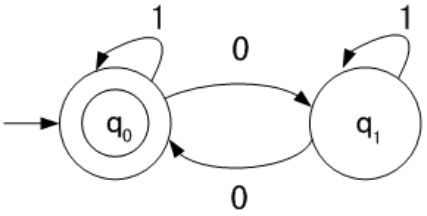

图5-10 状态转换图

在状态转换图中，末端没有状态连接的"→"所指向的节点为初态节点，而终态用双环圈表示，显然，该题中q0既是初态又是终态，那么该DFA显然能识别空串，当处于q0时输入0便转换到q1,而当处于q1时输入0又回到q0,并且在q0或q1状态连续输入n≥0个1仍回到原状态，因此，该DFA识别的串是包含偶数个0的二进制代码串，即在偶数个0之间或前后可插入任意个1的串，如ε、00、010、10101、1011011、1001101101.

将有限自动机M化为与其等价的正规式R的步骤为：

首先，在M的转换图上加进两个状态x和y,从x用标有ε的弧连接到M的所有初态节点，从M的所有终态节点用标有ε的弧连接到y,从而形成一个新的有穷自动机，记为M',它只有一个初态x和一个终态y,显然L（M）=L（M'），L（M）表示M所能接受的字符串集。

然后，逐步消去的所有节点，直到只剩下x和y为止，在消去节点的过程中，逐步用正规式来标记弧。其规则如图5-11所示。

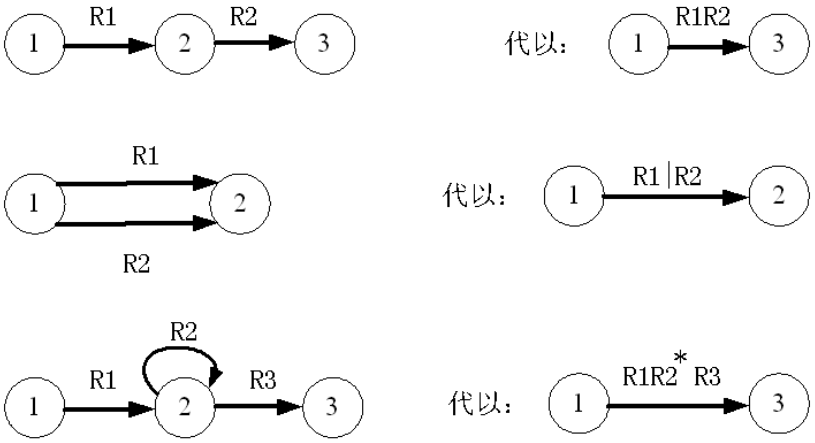

图5-11 DFA转正规式规则示意图

最后，从x到y的弧上标记的正规式即为所构造的正规式R.

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年05月20日

考点分析

第6章 计算机网络

计算机网络知识是软件设计师考试的一个必考点，每次考试的分数基本在5分左右。

6.1 考点分析

本节把历次考试中计算机网络方面的试题进行汇总，得出本章的考点，如表6-1所示。

表6-1 计算机网络试题知识点分布

考试时间	分数	考查知识点
10.11	4	100BASE-TX（1）、组播（2）、双绞线制作（1）
11.05	5	子网掩码（2）、远程管理（1）、TCP/IP端口（1）、带宽（1）
11.11	5	代理服务器（1）、ARP（1）、网络服务组件（1）、OSI参考模型（1）、OGSA标准（1）
12.05	5	多模光纤（1）、CDMA（2）、网络命令（1）、XML（1）
12.11	5	ARP（2）、802.11（1）、子网（2）
13.05	5	路由器概念（1）、ARP表（1）、C类网络（1）、网络命令（1）、DHCP（1）
13.11	5	FTP协议（1）、HTML语法（1）、POP3协议（1）、网络设计原则（1）、网络的需求分析（1）
14.05	6	UDP和HTTP协议（2）、电子邮件协议（1）、PING命令（2）、虚拟目录（1）

根据表6-1,我们可以得出计算机网络的考点主要有：

（1）网络体系结构：包括OSI参考模型、TCP/IP协议族中的各种协议。

（2）传输介质与设备：包括双绞线、光纤，各种网络使用的介质，以及常见的网络设备。

（3）组网技术：包括网络设计原则、网络的需求分析，以及各种接入技术。

（4）网络管理：包括代理服务器、常用的网络命令等。

（5）网络应用：包括各种常见的网络应用技术。

对这些知识点进行归类，然后按照重要程度进行排列，如表6-2所示。其中的星号（*）代表知识点的重要程度，星号越多，表示越重要。

在本章的后续内容中，我们将对这些知识点进行逐个讲解。