编译原理

田玲 教授、博导

lingtian@uestc.edu.cn

第四章 程序语言的设计

主要内容

1.语言的定义

- 语法
- 语义

2.文法

3.语言的设计

如何对高级语言给出明确的定

定义语言是否合法的一组规则

用以规定程序或其成分的含义

定义语言语法的形式化规则

介绍设计高级语言的一般知识和方法

第一节 语言的定义

定义 **程序设计语言**是用来描述计算机所执行的算法的形式表示

由两个部分组成：

- **语法**:用以构造程序及其成分的一组规则的集合
  - ✓生成的观点
  - ✓识别的观点
- **语义**:用以规定语法规则的集合
  - ✓指称语义学、操作语义学、代数语义学、公理语义学

用一组规则生成出一个语言；

用一个机制来识别一个语言；

第一节 语言的定义

- 语法

① 几个术语

- ✓ **字母表**:语言允许使用字符的集合,其元素称为字符；由字 号。
- ✓ **字汇表**:由符号组成的集合，其元素称为字
- ✓ **词法规则**:规定 的有效符号
- ✓ **语法规则**:规定 提供句子)

例如：26个大小写英文字母，10个数字，各种运算符号等等……

例如：for,while,procedure等各种关键 标识符，数字串等等

例如：算术表达式可以是单个的数字串，标识符，或由两个算术表达式经过算术运算构成；

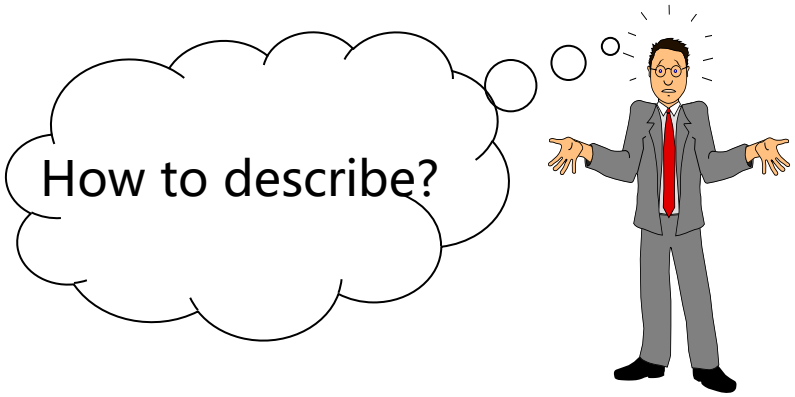
第一节 语言的定义

❑ 语法

② 生成的观点

例：一个简单英语句子如何按照生成的观点来描述它的语法：

I/Students study/run.



解决办法

<句子>→<主语><谓语>

<主语>→I|Students

<谓语>→ |run

注意：描述以上规则的符号系统称为巴科斯-诺尔范式，即通常的BNF（Backus-Naur Form）。

第一节 语言的定义

❑ 以上文法的形式描述

在形式语言中，上述例子可写成文法

G=(N,T,P,S)其中

- ✓ N={<句子>,<主语>,<谓语>}
- ✓ T={I,students,study,run}
- ✓ P={<句子>→<主语><谓语>,<主语>→I|Students,<谓语>→study|run}
- ✓ S=<句子>

非终结符的集合

终结符的集合

产生式的集合

文法的开始符号

例1：标识符的文法

<标识符>→<字母>
<标识符>→<标识符><字母>
<标识符>→<标识符><数字>
<字母>→A|...|Z|a|...|z
<数字>→0|...|9

例2：表达式的文法

<表达式>→<标识符>
<表达式>→(<表达式>)
<表达式>→<表达式><运算符><表达式>
<运算符>→+|-|*|/

第一节 语言的定义

- 语法

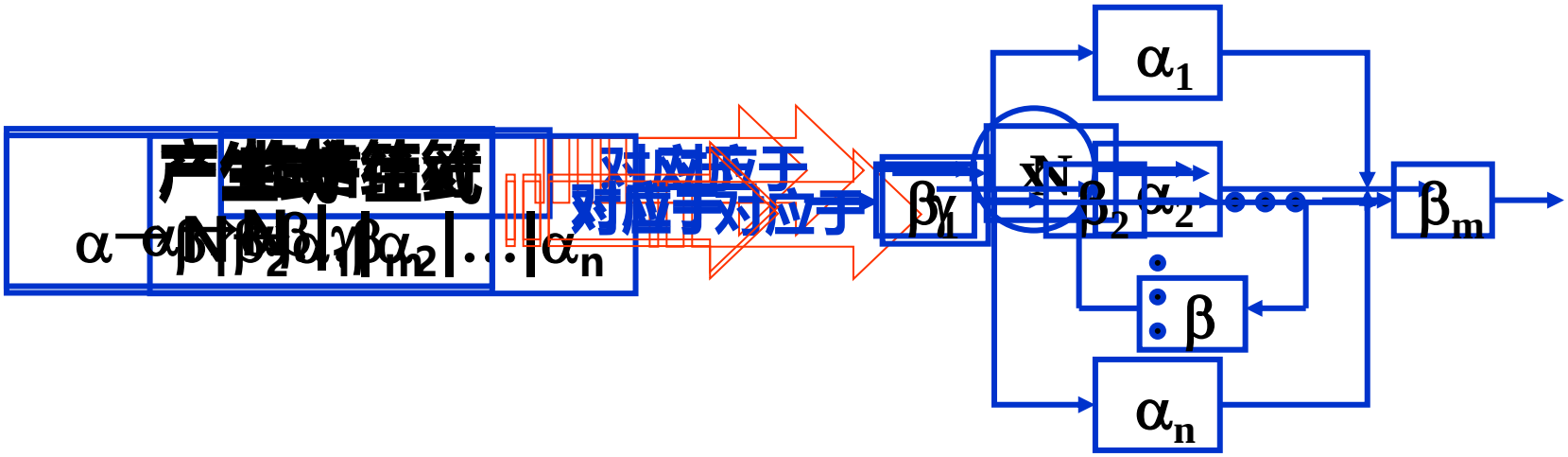
③ 识别的观点：**语法图**

语法图的定义

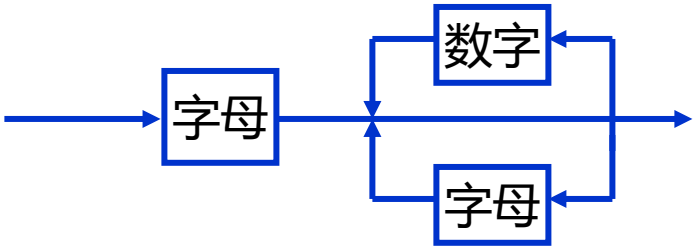
每一非终结符N连同相应的产生式

$N \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$

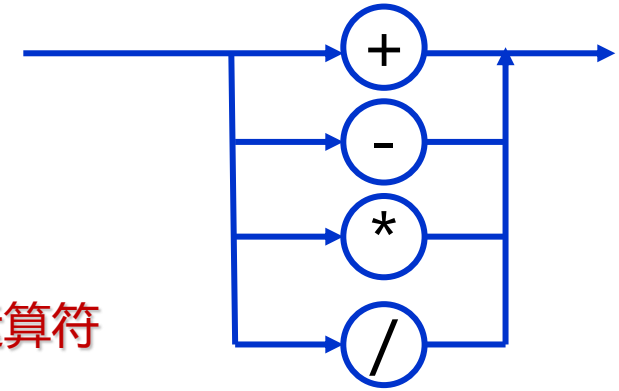
对应一个语法图；具体对应如下：



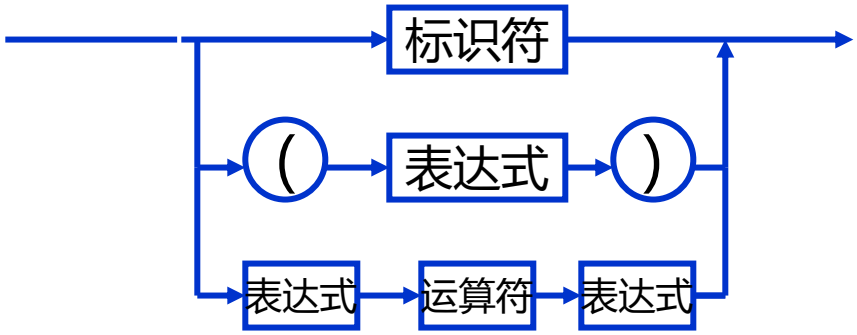
第一节 语言的定义



标识符

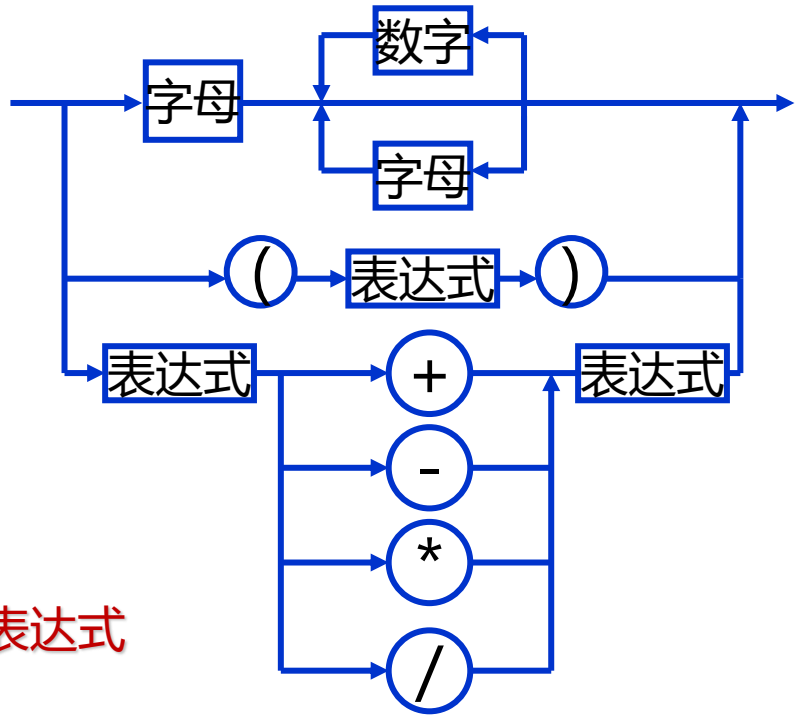


运算符



表达式

语法图的一些例子



表达式

第一节 语言的定义

- **识别原则：**

  若一个终结符序列是合法的，那么,必须从语法图的入口边通过语法图而到达出口边，而且在通过的过程中,恰恰能识别该终结符序列。具体如下：

  ①终结符框

  ②非终结符框

  ③分支

  ④回溯

  标识的终结符与被识别的终结符刚好符合；

  由该非终结符的语法图来识别；

  若遇到分支，则任选一分支来识别；

  若一个分支识别不成功，则选另一分支来识别；

注意：语法描述的用途

① 表达语言设计者的意图和设计目标;

② 指导语言的使用者如何写一个正确的程序

③ 指导语言的实现者如何编写一个语法检查程序来识别所有合法的程序。

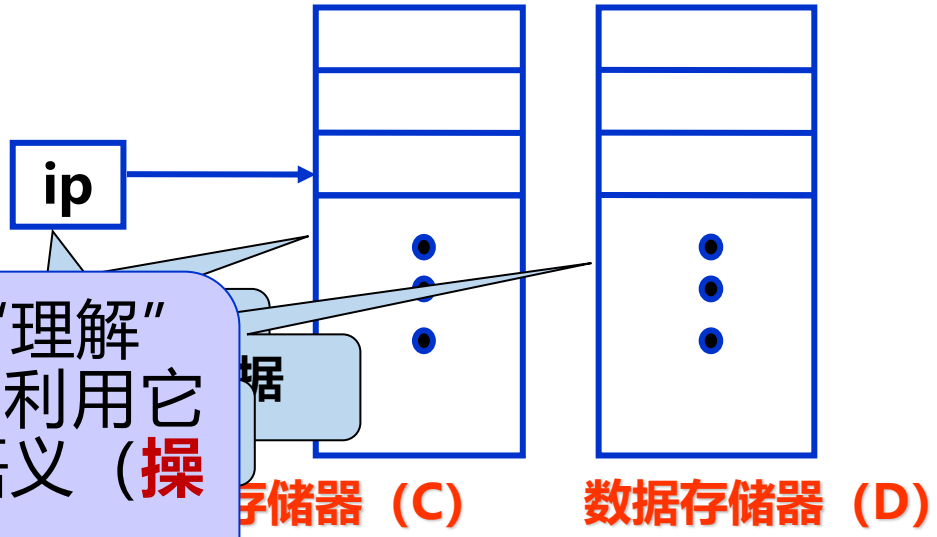
第一节 语言的定义

❑ 语义

定义 语言的**语义**定义语言合法句子的含义，即句子的作用和意义。

注意：虽然目前有很多种描述语言的语义的形式化方法，但目前无普遍接受的典型形式描述工具。

例：利用GAM抽象机来定义和理解语言结构的语义，它的简单结构如左图。



假定我们已经“知道”和“理解”了GAM上的语义，就可以利用它的操作来定义高级语言的语义（**操作语义**）。

第二节 文法

定义

文法G定义成一个四元式：

$G=(V_T,V_N,S,P)$

其中

- ✓ V_T是非终结符的集合
- ✓ V_N是终结符的集合
- ✓ $S\in V_N$是开始符号
- ✓ P是产生式的非空有限集

产生式一般写为$\alpha\rightarrow\beta$，其中$\alpha\in V^*V_NV^*$，$\beta\in V^*$，$V=V_N\cup V_T$

例：

$G_0=(V_T,V_N,S,P)$,其中

$V_T=\{+,*,(,),i\}$

$V_N=\{E,T,F\}$

$S=E$

$P=\{E\rightarrow E+T|T$

$T\rightarrow T*F|F$

$F\rightarrow(E)|i\}$

注意：多个产生式

$\alpha\rightarrow\beta_1$

$\alpha\rightarrow\beta_2$

…

$\alpha\rightarrow\beta_n$

一般缩写成$\alpha\rightarrow\beta_1|\ldots|\beta_n$

第二节 文法

❑ 文法的分类

等价定义：要求每个产生式均为αAβ→αωβ形式，其中A为非终结符

① **0型文法**:

- ✓ 产生式形如α→β， $\alpha \in V^*V_NV^*$，$\beta \in V^*$
- ✓ 又称为短语结构文法
- ✓ 0型文法产生的语言称为0型语言
- ✓ 0型文法的能力相当于图灵机（turing machine)
- ✓ 递归可枚举

② **1型文法**:

- ✓ 产生式形如α→β，$|\alpha| \le |\beta|$，
- ✓ S→ε除外，S不得出现在任何产生式右部；
- ✓ 又称为**上下文相关文法；**
- ✓ 产生的语言称为1型语言，也称为上下文相关语言
- ✓ 生成语言一定是递归集，递归集不一

③ **2型文法**:

- ✓ 产生式形如A→β，$A \in V_N$，$\beta \in V^*$
- ✓ 又称为**上下文无关文法**
- ✓ 2型文法产生的语言称为2型语言，也称为上下文无关语言（CFL）
- ✓ 2型文法的能力相当于下推自动机
- ✓ 通常强制式语言属于2型语言

③ **3型文法**:

- ✓ 产生式形如A→αB，或A→α，A，$B \in V_N$，$\alpha \in V_T^*$
- ✓ 又称为**正则文法(右线性文法)**
- ✓ 3型文法产生的语言称为3型语言，也称为正则语言（正则集）
- ✓ 3型文法的能力相当于有限自动机
- ✓ 3型语言是2型语言的一个子集

第二节 文法

❑ 文法产生的语言

定义

设文法$G=(V_T,V_N,S,P)$，$\alpha\beta\gamma\in V^*$

① 如果$\beta\to\delta\in P$，则$\alpha\beta\gamma$称**直接推导**出$\alpha\delta\gamma$，记为$\alpha\beta\gamma\Rightarrow\alpha\delta\gamma$；其逆过程称为归约；

② 如果$\alpha_1\Rightarrow\alpha_2\ldots\Rightarrow\alpha_n$，则称记为$\alpha_1$ **推导**出α_n，记为$\alpha_1\overset{+}{\Rightarrow}\alpha_n$；

③ 如果$\alpha_1\overset{0}{\Rightarrow}\alpha_n$，或$\alpha_1\overset{+}{\Rightarrow}\alpha_n$，则记为$\alpha_1\overset{*}{\Rightarrow}\alpha_n$；

例：

已知G(E)={E→E+E | E*E | (E) | i}, i+i*i的

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

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

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

最右推导（规范推导），逆过程称为最左归约（规范归约）

最左推导，逆过程称为最右归约

该句型的另一个规范推导

第二节 文法

定义

设文法$G=(V_T,V_N,S,P)$，$\alpha,\beta\in V^*$，

① 如果$S\overset{*}{\Rightarrow}\alpha$，$\alpha\in V^*$，则$\alpha$为G的一个**句型**；

② 如果$S\overset{*}{\Rightarrow}\alpha$，$\alpha\in V_T^*$，则$\alpha$为G的一个**句子**

③ 文法G的所有句子的集合称为G产生的**语言**，记为L（G),即：

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

例1：文法$G=\{S\rightarrow aS|b\}$产生的语言L（G）$=\{a^nb|n\geq 0\}$;

例2：$G=\{S\rightarrow aS \mid aP$

$P\rightarrow bP \mid bQ$

$Q\rightarrow cQ \mid c\}$

它产生语言$L(G)=\{a^ib^jc^k \mid i,j,k\geq 0\}$

例3：$G=\{S\rightarrow aSPQ \mid abQ$

$QP\rightarrow PQ$

$bP\rightarrow bb$

$bQ\rightarrow bc$

$cQ\rightarrow cc\}$

它产生语言$L(G)=\{ a^ib^ic^i \mid i\geq 1\}$

一个上下文相关文法

第二节 文法

❑ 语法树

定义 **语法树**（或推导

推导过程的树

① 每个

文法

② 如果

右有

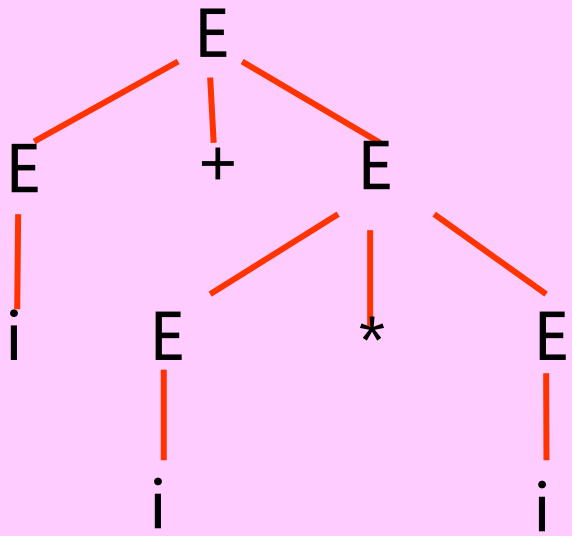
A→

③ 如果

，且

的

到



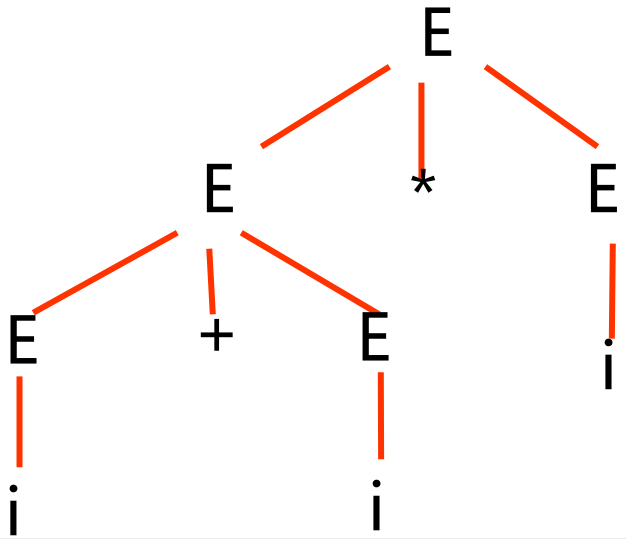
例：

$G_1=\{E\rightarrow E+E|E*E|(E)|i\}$

句子i+i*i的语法树下：

第二节 文法

注意：i+i*i除了上面的语法树之外，还存在不同的语法树。



文法G_0={E→E+T|T
T→T*F|F
F→(E)|i}
和文法G_1={E→E+E|E*E|(E)|i}
产生相同的语言（请大家验证），但G_1是二义文法，而G_0不是。

定义

如果一个文法存在某个句子有多于一个的语法树，则称这个文法是**二义文法**。

几个结论

① 一个二义文法产生的语言不一定是二义语言；

② 二义性的问题是不可判定的；

③ 存在先天二义语言；即，每个产生它的文法都是二义的；

第三节 语言的设计

- 主要目的：给出设计一个强制式语言的入门知识和方法。

- 表达式的设计

强制式语言对数据[illegible]达式中完成，表达式通常[illegible]

  - ✓ 逻辑表达式 (log[illegible]
  - ✓ 关系表达式 (ration[illegible]pression)
  - ✓ 算术表达式 (arithmetic expression)

处理对象为逻辑数据，如布尔变量[illegible]

用来判定某个条件成立或不成立，处理对象通常为算术表达式；运算为关[illegible]

处理对象是各种整数、实数；运算为算术运算；

第三节 语言的设计

❑ 逻辑表达式

运算符 ¬（非）、∧（与）、∨（或）

要求

满足优先次序； ¬ > ∧ > ∨
满足左结合；

❑逻辑表达式的BNF范式

<逻辑表达式>→<布尔常量>|<布尔变量>|<关系表达式>
|¬<逻辑表达式>
|<逻辑表达式>∨<逻辑表达式>
|<逻辑表达式>∧<逻辑表达式>
<布尔常量> →true|false
<布尔变量> →<标识符>

第三节 语言的设计

❑ 关系表达式

运算符 < （小于），<= （小于等于），> （大于），
>= （大于等于），= （等于），<> （不等于）

要求 运算符之间没有优先次序
运算符没有重载

❑关系表达式的BNF范式

<关系表达式>→<关系表达式><关系运算符><关系表达式>

<关系运算符>→<|<=|>|>=|=|<>

注意：不同的语言之间，关系运算符的记号可能有所不同。

第三节 语言的设计

❑ 算术表达式

运算符 各种算术运算，常见的有+(加)，-(减)，*(乘)，/(除)等

要求 运算符的优先次序，*,/ ≥ +, -；
同优先级算符左结合；

❑算术表达式的BNF范式

<算术表达式>→<算术表达式>+<项>|<算术表达式>-<项>|-<项>|<项>

<项>→

<因子>

<变量>

<常量>

注意，文法
<算术表达式>→<常量>|<变量>|（<算术表达式>）
|<算术表达式><算术运算符><算术表达>
是二义文法，只能在某些特点场合使用；

第三节 语言的设计

❑语句的设计

强制式语言是面向语句的语言，能过语句描述问题的求解过程。语句通常分为：

- ✓ 说明语句（declare statement）
- ✓ 控制语句（execute statement）

不生成目标代码，用来告诉编译程序一些实体的属性；

生成目标代码；

❑ 说明语句

说明语句主要包括变量说明和类型说明；

<类型说明>→type <类型名>=<用户定义类型>
<类型名>→<标识符>
<用户定义类型> （从略）

<变量>→<标识符>
<类型>→integer|real|char|boolean

第三节 语言的设计

❑执行语句

执行语句主要包括赋值语句、控制语句和复合语句；

1. 赋值语句

<赋值语句> → <变量>:=<表达式>

2. 控制语句

- ✓ 顺序、控制和循环
- ✓ 语句的选择：在多样性和效率之间折衷
- ✓ 语句的结束符：两个含义，控制执行顺序和语句结束

3. 复合语句（compound statement）

<复合语句> → begin <语句表> end

<语句表> → <语句>|<语句表>;<语句>

第三节 语言的设计

❑程序单元的设计

程序单元是程序可以独立调用的成分，在设计时要考虑下面的问题：

- ✓ 程序单元的局部环境如何定义
- ✓ 程序单元的头尾标识
- ✓ 程序单元的名字及参数如何定义
- ✓ 程序单元如何被调用及参数如何传递

❑程序单元的BNF范式

<程序单元> → <程序单元关键字><程序单元名字>(<形参表>)；<程序单元体>

<程序单元关

<程序单元名

<形参表> → <形参>

>;<形参>

注意，定义形参，要说明其类型；说明与实参的绑定方式；还要说明参数传递方式（见其他章节）

第三节 语言的设计

❑程序单元的BNF范式（续）

<程序单元体> → begin<说明部分>;<执行部分> end

<说明部分> → <说明语句表>

<说明语句表> → <说明语句>|<说明语句表>;<说明语句>

<执行部分> → <执行语句表>

<执行语句表> → <执行语句>|<执行语句表>;<执行语句>

例：ALGOL 68中分程序的定义

<分程序> → begin <说明部分>;<执行部分>end

与上类似

<说明部分> → <变量说明表>;<数组说明表>;<过程说明表>;<分程序表>

<变量说明表> → <变量说明>|<变量说明表>;<变量说明>

<数组说明表> → <数组说明>|<数组说明表>;<数组说明>

<过程说明表> → <过程说明>|<过程说明表>;<过程说明>

<分程序表> → <分程序>|<分程序表>;<分程序>

第三节 语言的设计

❑程序的设计

程序通常由一个关键字，后跟程序名，参数表以及程序体构成，即：

<程序> → program<程序名>(<参数表>);程序体

不同的语言有细微的差别

第四节 语言设计实例

- 主要内容：给出一个极小的强制式语言的设计，它面向阶乘n!的求解，最后用该语言写出求阶乘的程序

问题：阶乘n!的求解

if n<=0 then F:=1 else F:=n*F(n-1)

- 该语言的BNF范式

<程序> → <分程序>

<分程序> → begin <说明语句表>;<执行语句表> end

<说明语句表> → <说明语句>|<说明语句表>;<说明语句>

<说明语句> → <变量说明> <函数说明>

<变量说明> → integer <变量>

注意，只允许无符号整形变量

第四节 语言设计实例

- 该语言的BNF范式（续）

<变量> → <标识符>
<标识符> → <字母>|<标识符><字母>|<标识符><数字>
<字母> → a|b|……|y|z
<数字> → 0|1|……|9
<函数说明> → integer function<标识符>(<参数>);<函数体>
<参数> → <变量>
<函数体> → begin <说明
<执行语句表> → <
<执行语句> →
句
<读语句> → read (<
<写语句> → write (<变量>)

函数可递归调用；返回值为整数，存放在标识符（函数名）中

从键盘上读入数据放入变量

把变量中的值输出到屏幕上

第四节 语言设计实例

- 该语言的BNF范式（续）

<赋值语句> → <变量>:=<算术表达式>

<算术

<项>

<因子

<常数

<无符

<条件

<条件

<关系

例：计算阶乘n!的程序如下

```
begin
  integer k;
  integer function F(n);
    begin
      integer n;
      if n<= 0 then F:=1;
      else F:=n*F(n-1);
    end
  read(m);
  k:=F(m);
  write(k);
end
```

常的

算术表

第五节 一些设计准则

- **可写性**：语言应提供一些构造使程序员方便地完成程序设计，让程序员把注意力集中在问题的求解上，而不必把注意力集中在表达求解工具上；可写性主要表现在**简单性**，**可表达性**，**正交性**和**准确性**方面

语言的语法及语义应该是精确的，无二义的。

- **可读性**：程序语言应该容易阅读；可读性与可写性相关，影响到语言的可修改性和可维护性

例如，Ada语言中的数据抽象、信息屏蔽、模块的独立性、可见部分与实现部分的分离，私有类型、派生类型等都是为了提高程序的可靠性；

- **可靠性**：可靠性[illegible]件系统正常工作的能力；一个语言应该使得[illegible]编制的软件具有高的可靠性；语言的很多**构造**是为了提高可靠性

作业

1. 文法G：S→xSx|y所识别的语言是_____。

2. 给出生成下述语言的上下文无关文法：

（1）$\{a^n b^n a^m b^m | n,m>=0\}$

（2）$\{1^n 0^m 1^m 0^n | n,m>=0\}$

3. 文法G=({A,B,S},{a,b,c},P,S)

其中P为：S->Ac|AB

A->ab

B->bc|c

是二义的吗？为什么？

4. 文法G[E]为：

E → T|E+T|E-T

T → F|T*F|T/F

F →(E)|i

证明E+T*F是它的一个句型，指出这个句型的所有短语、直接短语和句柄。