# 第 8 章 程序设计

## 8.1 过程

### ✧ 过程与复合表达式

简单地说，在 *Mathematica* 中的一个过程是用分号隔开的表达式序列，一个表达式序列也称为一个复合表达式。

在函数定义中如果要用一串命令，即一个复合表达式完成计算，请将这一串命令用圆括号括起来，并以最后一个表达式的值作为函数值。例，

```
In[1]:= gun[x_]:=(x=2 x; y = 3 x)
In[2]:={fun[3], gun[3]}
Out[4]={6,9}
```

### ✧ Modules 和局部变量

一般形式：

**Module [**{局部变量表}，body**]**

{局部变量表}中可说明 0 个或多个局部变量，说明变量时只列出变量名，不需对变量进行类型说明。局部变量名之间用逗号分隔，并可在说明时赋以初始值，如 In[3] 中所示。

Module 在运行前先完成初始值的定义和赋值，再依次计算 Module 中复合表达式 body 中的表达式，并以最后一个表达式的值作为 Module 结构的值，如 In[2] 中所示。也就是说当表达式为复合表达式时，最后一个表达式的值是该复合表达式的值。我们知道 *Mathematica* 不输出以分号标识的表达式的计算结果，如果需要输出中间结果可用 Print 函数.

### ✧ With 和局部常量

用 Module 建立局部变量同时也能给局部变量定义初始值。有时我们并不需要局部变量只需要局部常量。在 *Mathematica* 中用 With 结构定义局部常量。

**With [**{x=x0, y=y0, …}, body **]**

其中：定义局部常量 x, y, …的值为 x0,y0,…，body 是复合表达式。

可以认为 With 结构对 body 使用变量替换运算符号“/.”。 With 是一类特殊形式的 Module，它的局部变量只赋一次值。请注意区分在 Function, With 和 Module 中的 x 的含义。

Function [{x, …}, body ]　　x 是局部参量

With [{x=x0, y=y0, …}, body ]　　x 是局部常数

Module [{x,　…}, body ]　　x 是局部变量

```
In[1]:=u =11; w[x_]:= With[{u =(x+1)^2},u+u^2]
In[2]:= w[a]
Out[1]=(1+a)^2 +(1+a)^4
```

```
In[2]:=Module[{t=8}, With[{t = 9},t^2]]
Out[2]=81
```

✧ Block 和局部变量

Block 的一般形式：

**Block [**{变量名}，body**]**

变量名之间用逗号分隔，并可在说明时赋以初始值，如 In[8] 中所示。body 为一个复合表达式，表达式之间用分号相隔，并以最后一个表达式的值作为 Block 结构的值。

Block 的常用形式：

**Block [**{x, y,…}, body **]**　用 x、y 的局部值运算 body

**Block [**{x = x0 y = y0,…}, body **]**　给 x、y 赋以初始值再运行 body

Block 中的 x 是局部变量，在 Block 内的任何局部变量值的有效范围都在所定义的块内。Block 能有效地建立程序的小环境，在小环境中可以临时改变任何变量的值。*Mathematica* 在 Do, Sum, Table 等函数中有效地利用 Block 建立局部循环变量。例如，Sum [ t^2, {t, 10} ]中的变量 *t* 就是一个典型的局部变量。

Module [ vars, body ]　vars 词法作用域 (lexical scoping)

Block [ vars, body ]　vars 动态作用域 (dynamic scoping)

```
In[3]:=g[x_]:=Block[{t}, t=Inverse[x]; TableForm[t]]
In[4]:= s ={{1.,0,1},{0,1,2},{1,2,1}};
In[5]:= g[s]    (* 计算矩阵 s 的行列式和逆矩阵 *)
Out[5]//TableForm
                0.75      -0.5      0.25
                -0.5      0.        0.5
                0.25      0.5       -0.25
```

如果一个过程中没有局部变量说明，则可省略 Module 或 Block 的框架。

| *Mathematica* 的过程形式 | 说 明 |
|---|---|
| 表达式 1；表达式 2；…表达式 n | 由一串命令组成的过程（复合表达式 ） |
| Module [{局部变量表}，表达式] | 2.0 以上的版本 |
| With [{局部常量表}，表达式] | 2.0 以上的版本 |
| Block [{变量名}，表达式] | 1.2 以上的版本 |

表 8 - 1

例：随机形成元素在 [10，100] 之间的 n 阶矩阵，找出矩阵第 i 行的最大值、第 j 列的最小值。

```
In[1]:=jsa[n_,i_,j_]:=Module[{A},
  A =Table[Random[Integer,{10,100}],{n},{n}];
     m1 = Max[A[[i]]]; m2 = Min[A[[All,j]]];
    Print["m1=", m1, ", m2=", m2]]
```

```
In[2]:=jsa[5,2,3]
Out[2]= m1=87 ,  m2=50
```

## 8.2 条件控制结构

### ✧ If 语句

If 语句的结构与一般程序设计语言结构类似。由于 *Mathematica* 的逻辑表达式的值有三个：真（True）、假（False）和“非真非假”即无法判定。因此，相应的条件语句的转向也有三种情况。下列 If 结构的三种情况。

**If [**逻辑表达式, 表达式 1**]**

当逻辑表达式的值是真（True） 时，计算表达式 1，表达式 1 的值就是整个 If 结构的值。

**If [**逻辑表达式, 表达式 1, 表达式 2**]**

当逻辑表达式的值是 True 时,计算表达式 1，并将表达式 1 的值作为整个结构的值；当逻辑表达式的值是 False 时,计算表达式 2，并将表达式 2 的值作为整个结构的值。

**If [**逻辑表达式, 表达式 1, 表达式 2, 表达式 3**]**

当逻辑表达式的值是 True 时，转向计算表达式 1，当逻辑表达式的值是 False 时转向计算表达式 2，当逻辑表达式的值非 True 非 False 时，计算表达式 3，并将所计算表达式的值作为整个 if 结构的值。

```
In[1]:=f[x_,y_]:= If[x>0 && y>0,x+y,x-y]
In[2]:=f[3,3]
Out[2]=6
In[3]:= f[2,u]
Out[3]= f[2,u]    (*因没有给出 u 的值,Mathematica 无法判断 u>0*)
In[4]:=g[y_]:=If[y>0,"ABC","DEF","XYZ"]
In[5]:=g[z]
Out[5]=XYZ (*Z 没有赋值, 逻辑表达式 z>0 的值非 True 非 False*)
```

### ✧ Which 语句

Which 语句的一般形式：

**Which [**条件 1, 表达式 1, 条件 2, 表达式 2,…,条件 n, 表达式 n**]**

**Which [**条件 1, 表达式 1,…,条件 n, 表达式 n, True, 表达式**]**

依次计算条件 i，计算对应第一个条件为 True 的表达式的值，作为整个结构的值。如果所有条件的值都为 False，则整个结构的值是 Null ，用 True 作为 Which 的最后一个条件时，用于处理其它情况，相当于 C 语言中 Switch 语句中的 default 的作用。

例 ：计算

$$h(x)=\begin{cases} -x, & x<0 \\ \sin(x), & 0\le x<6 \\ x/2, & 16\le x<20 \\ 0, & \text{其它} \end{cases}$$

```
In[6]:=h[x_]:=Which[x<0,-x,x>=0 && x<6,Sin[x],
                    x>=16 && x<20,x/2,True,0]
In[7]:=h[-12]
Out[7]=12
In[8]:= {h[5],h[16.2],h[z]}
Out[8]={Sin[5],8.1,0}    (*z 未赋值，不满足前三个条件*)
In[9]:= k[x_]:= Which[x>1, u=1,x>2, v=2, x>3, w=3]
In[10]:= k[6]
Out[10]= 1    (*同时满足三个条件，执行第一个条件对应的表达式 *)
```

✧ Switch 语句

语句的一般形式：

**Switch [**expr, 模式 1, 表达式 1, 模式 2, 表达式 2,…**]**

将表达式 expr 的值与模式 1，模式 2， …，依次做比较，给出第一个与 expr 匹配的模式 i 对应的表达式 i 的值。若没有匹配的模式，则整个结构的值为 Null。.

```
In[11]:= g[x_]:=Switch[Mod[x,3],0,a,1,b,2,c]
In[12]:={g[7],g[8],g[9]}
Out[12]={b,c,a}
```

## 8.3 循环控制结构

✧ Do 语句

Do 语句的一般形式为：**Do [**循环体，{循环范围}**]**

Do 语句有下列形式：

| | |
|---|---|
| Do [expr,{i,io,i1,s}] | 循环变量 i 从 i0 到 i1 ,每次 i 增加 s ,计算表达式 expr. |
| Do [expr,{i,i1}] | 同上,当循环初值 i0=1 ,步长 s=1 时可省略不写 |
| Do [expr,{n}] | 对表达式 expr 计算 n 次 |
| Do[expr,{i,i0,i1,is},{j,j0,j1,js}] | i 从 i0 到 i1 按步长 is 递增 ； 对每个 i , j 从 j0 到 j1 按步长 js 递增，计算表达式 expr |

```
In[1]:=Do[Print[i^3],{i,3}]
In[2]:=t=x; Do[t=1/(1+k t),{k,2,6,2}];t
```

Out[2]=
$$\cfrac{1}{1+\cfrac{6}{1+\cfrac{4}{1+2x}}}$$

```
In[3]:=Do[Print[{i,j}],{i,2},{j,i}]
```

✧ While 语句

一般形式： **While [**条件，循环体**]**

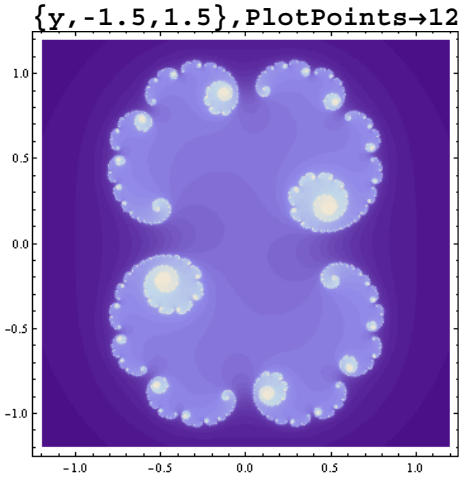
```
In[4]:= n = 19; While[(n=Floor[n/2]) != 0,Print[n]]
```

```
In[5]:= While[u>1, Print[u]]  (*毫无反映,一次也不做，因为 u 未赋值*)
In[6]:= u = 2 ; While[u>1,Print[u]]
```

屏幕上不停地显示 2,按“Alt+,”中断程序计算，返回交互式状态。具体操作请看 8.5.4。

例：用当型语句 While 和密度函数（DensityPlot）画 Julia 分形图形。

```
In[1]:=julia[x_,y_,lim_,cx_,cy_]:=Module[{z,ct=0},z=x+I y;
  While[Abs[z]<2.0 && ct <lim,++ct; z=z*z+(cx+I cy)];Return[ct]]
In[2]:=DensityPlot[julia[x,y,50,0.27334,0.00742],{x,-1.5,1.5},
       {y,-1.5,1.5},PlotPoints→120]
```




✧ For 循环

For 语句的一般形式：

**For [**初始值，条件，修正循环变量，循环体**]**

```
In[7]:= For[i=1;t=x,i^2 <10,i++,t=t^2+1;Print[t]]
```

$1+x^2$

$1+\left(1+x^2\right)^2$

$1+\left(1+\left(1+x^2\right)^2\right)^2$

*Mathematica* 中的 For 和 While 和 C 语言中的 For 和 While 的工作方式大致相同，也许你已经发现也有不同之处，逗号和分号的作用在 *Mathematica* 中和在 C 语言中正好相

反。

下列操作方式会给你在循环结构中的运转代来一些方便。

| 函 数 | 说 明 |
|---|---|
| k++ | k 的值增加 1 |
| ++k | 先增加 I |
| k–– | k 的值减少 1 |
| ––k | 先减少 1 |
| {x, y}={y, x} | 交换 x 和 y 的值 |
| PrependTo[list, elem] | 将 elem 放到表 list 的最前面位置 |
| AppendTo [list, elem ] | 将 elem 放到表 list 的最后面位置 |

### ✧ 重复运用函数的方法

**Nest [**f, expr, n**]** 对表达式 expr 作用 f 函数 n 次

**NestList [**f, expr, n**]** 从表达式 expr 作用 f 函数 n 次，并给出列表结果

**FixedPoint [**f, expr**]** 从表达式 expr 开始，重复运用 f 函数，直到结果不变

例如：

```
In[1]:=Nest[f,x,5]
Out[1]= f[f[f[f[f[x]]]]]
In[2]:=FixedPoint[Function[x,Print[x];Floor[x/4]],9]
```

计算 $\sqrt{3}$ 的牛顿迭代法公式为：$x_{k+1}=\frac{1}{2}(x_k+\frac{3}{x_k})$

```
In[3]:=newton[x_] := N[1/2( x + 3/x )]
```

以 1.0 为初值, 做 5 次迭代并输出计算结果。

```
In[4]:=NestList[newton,1.0,5]
Out[4]= {1.,2.,1.75,1.73214,1.73205,1.73205}
```

### ✧ 过程程序和函数程序

从程序设计方法的角度看，在 Fortran 和 C 等高级语言中，编制的程序是过程程序（Procedural Program）。编制程序中总要精心地设置一些局部变量，循环地控制结构和子程序等，它的优点是有利于充分发挥个人的智慧和才能。

在符号计算系统中，编制的程序是以函数程序（Functional Programming）方式为主，编程中重点放在分析要解决问题的方法和算法，编制程序就是调用和组合相应的系统函数，使用函数而不用编程或少量编程，有效地减少编程中细微而又繁琐的工作，简化或改变了递归的形式。如果你已经习惯在 C 语言中的编程方法，在符号计算系统中，也有各类语句供你使用，仍然可用“过程程序”的思想方法编制程序。

## 8.4 转向控制

✧ **复合表达式内的控制转向**

**Label [**name**]**　用标识符 name 标出复合表达式中的一个位置

**Goto [**name**]**　转向当前过程中 Label [name]位置后执行

通常系统在复合表达式中按表达式的顺序依次求值，遇到 Goto[name] 时，转向处于同一复合表达式 Label [name]的位置处，从那里继续运算。

✧ **退出循环结构**

在 *Mathematica* 中退出循环结构可用下列函数：

| 函 数 | 说 明 |
|---|---|
| Return [expr] | 退出函数中的所有过程和循环,返回 expr 的值 |
| Break [ ] | 结束本层循环,并以 Null 为结构的值 |
| Continue [ ] | 转向本层 For 或 While 结构中的下一次循环 |
| Throw [expr] | 返回 expr 作为最近的外层的 Catch 的值 |

表 8 - 3

可以看到 Return、Break 和 Continue 在 *Mathematica* 中的工作方式与 C 语言中的相同。

```
In[1]:= f [y_,x_]:= Block[{t},t=D[y,x];If [t==0, Return ["***"]];
                    t=t^2+1; Return[t]]
In[2]:=f[6,x]
Out[2]= ***
In[3]:=t=1; Do[t*=k;Print[t];If[t >19,Break[ ]],{k,10}]
In[4]:=t=1;Do[t*=k ; Print[t] ;
            If[k < 3, Continue[ ]] ;t +=2,{k,5}]
```

## 8.5 构造程序包

### 8.5.1 给函数定义信息

在定义函数时可同时定义与函数相关的信息，就象 *Mathematica* 的内部函数所做的那样。例如：

```
In[1]:=Sqrt[a,b]
      Sqrt :: arget: Sqrt called with 2 argument.
Out[1]= Sqrt[a,b]
```

*Mathematica* 显示错误的原因是在求根函数中用了 2 个变量

```
In[2]:= f :: overflow ="Factorial too large."
```

```
Out[2]= Factorial too large.      （* 自己定义信息 *）
In[3]:= f[x_Integer]:=If[x>10,Message[f::overflow];infinity,x!]
In[4]:= f[5]
Out[4]=120
In[5]:= f[16]
        f :: overflow : Factorial too large.
Out[5]= infinity    (* x> 15,显示信息 overflow *)
In[6]:= Message[f]
Out[6]= f :: overflow -> Factorial too large.
In[7]:= Off[f::overflow]  (*关闭 信息 *)
```

下列有关信息操作的命令和意义 ：

| 命 令 | 说 明 |
|---|---|
| s :: tag =String | 定义一条信息 |
| Message [s :: tag] | 显示一条信息 |
| Message [s] | 显示与 S 相关的信息 |
| Off [s :: tag] | 关闭 S 的 tag 信息 |

### 8.5.2 程序中的注释

任何语言都有它的注释语句方式。言精意明的注释有助于检查程序，有利于正确地理解程序。

在 *Mathematica* 中用 (*注释内容*) 的形式，在语句中加以注释。注释可以加在程序的任何地方，注释中的内容可为任何字符，包括汉字、空字符和换行符等，注释中还允许嵌套注释。

程序在运行中跳过注释部分，并将注释语句看成一个空格。

### 8.5.3 程序包结构

下面列出 *Mathematica* 的程序包结构的一般形式。与 C 语言等高级语言不同的是：除了包的主体以外，程序包结构中的很多项都可以省略。例如：可以没有 BeginPackage [“程序包名”]，也可以没有 f :: usage=“说明”或 Begin ["Private`"]。在程序中，如果不设 BeginPackage [“程序包名”]，也就没有 EndPackage [ ]这一项。用样 Begin [" Private` "] 与 End [ ]也是同时出现的。

```
BeginPackage["程序包名"]
f :: usage=“说明”，…        (* 引入作为输出的目标 *)
Begin["Private` "]           (*开始程序包的私有上、下文 *)
f [变量] = 表达式            (*包的主体 *)
```

```
...
End[ ]                          (* 结束自身的上下文 *)
EndPackage[ ]                   (*程序包结束标志，并将该程序包放到全局上下文
                                  路径的最前面 *)
```

### 8.5.4 程序执行的中断和退出

✧ **中断程序运行**

在程序运行中遇到运行时间过长或死循环需要人工中断，按热键“Alt+ ，”或单击 Kernel 菜单中 Interrupt Evaluation 命令项。

✧ **退出程序运行**

在程序运行中，按热键“Alt+ .”或单击 Kernel 菜单中 Abort Evaluation 命令项，系统则退出全部表达式计算，并返回值$Aborted。

## 8.6 程序实例

例 6：计算一组数据的算术平均值、几何平均值、中差、方差和标准偏差。

程序包设计如下：

```
BeginPackage["Statistics`"]
Mean[list_List]:=Apply[Plus,list]/ Length[list]
    (* 计算数据 list 的算术平均值 *)
GeometricMean[list_List]:=Apply[Times, list]^(1/Length[list])
    (* 计算数据 list 的几何平均值 *)
Median[list_List]:=Block[{s1,len},
              len= Length[list]; s1= Sort [list];
              If [OddQ[Length[s1]],s1[{len/2}],
                              (s1[[len/2]]+ s1[[len/2+1]])/2 ]
                 ]     (* 计算数据 list 的中差 *)
Variance[list_List]:= Mean[(list–Mean[list])^2]
                      (* 计算数据 list 的方差 *)
`Range[list_List]:=Apply[Max,list] –Apply[Min,list]
                    (*  计算数据的标准偏差  *)
(* `Range *Hides System`Range* *)
EndPackage[ ]
In[1]:= << Statistics.m
In[2]:= data =Table[Random[ ],{10}];
In[3]:= Mean[data]
Out[3]= 0.359058
```

```
In[4]:= GemetricMean[data]
Out[4]= 0.257275
In[5]:= Range[data]          (* Statistics`包中的 Range 函数 *)
Out[5]= 0.684207
In[6]:= System`Range[6]      (* 系统中的 Range 函数 *)
Out[5]= {1,2,3,4,5,6}
```

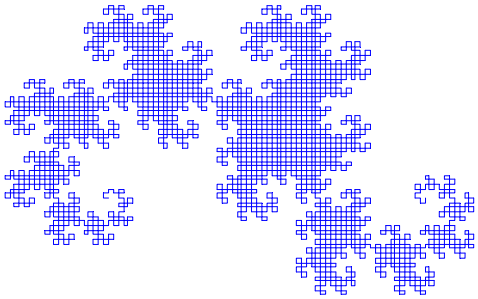
例 7：用龙格库塔方法解常微分方程

$$\begin{cases} \dfrac{dy}{dx} = f(x,y) \\ y(x_0) = y_0 \end{cases} \qquad y_{n+1} = y_n + (k1 + 3k2 + 3k3 + k4)/8$$

$$\begin{cases} k1 = hf(x_n, y_n) \\ k2 = hf(x_n + h/3, y_n + k1/3) \\ k3 = hf(x_n + 2h/3, y_n - k1/3 + k2) \\ k4 = hf(x_n + h, y_n + k1 - k2 + k3) \end{cases}$$

例8：绘制三龙曲线

```
In[1]:=dragon[x1_, y1_, x2_, y2_, n_]:= Module[{},
     If[n > 0,dragon[x1,y1,(x1+x2+y1-y2)/2, (y1+y2-x1+x2)/2,n - 1];
  dragon[x2,y2,(x1+x2 +y1-y2)/2,(y1+y2 -x1+x2)/2,n-1],
     (*else*)res = Append[res,{RGBColor[0,0,1],
          Line[{{x1,y1}, {(x1 + x2 + y1 - y2)/2, (y1 + y2 - x1 + x2)/2},
          {x2, y2}}]}]] (*end if*)] (*end Module*)
In[2]:=res = {};
      Show[Graphics[dragon[0,0,1,0,11]]]
```



例 9： 模拟地月日三球模型。