第3章 程序构造方法

程序构造的最基本手段为抽象和组合。抽象指将物理的与、或、非门抽象为与、或、非操作。与、或、非操作被组合为算术运算、逻辑运算和关系运算这样的计算机基本指令。完成某一任务的指令序列组合为程序。组合中的自相似性定义为递归，组合中的非自相似重复定义为迭代，为实现某一逻辑功能的组合打包定义为函数(前缀表达式的新运算符)。

3.1 导学导教

3.1.1 内容导学

本章内容导学图如图 3-1 所示。

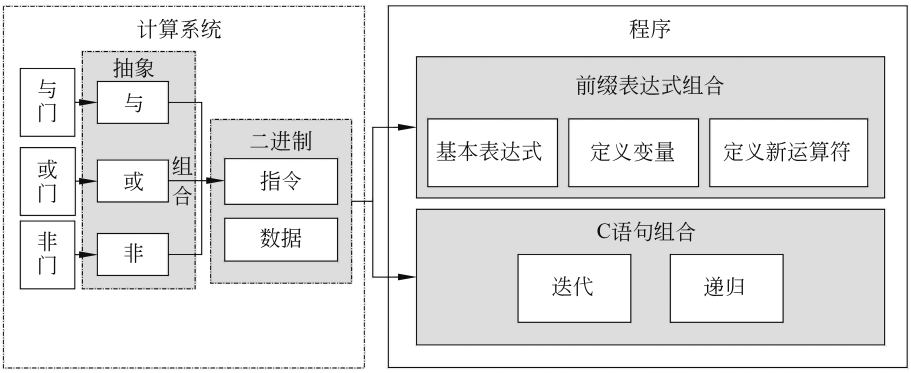

图 3-1 程序构造方法内容导学图

3.1.2 教学目标

1. 知识目标

理解运用组合与抽象构造计算系统的基本思想，掌握运用前缀表达式构造语法和计算过程，掌握运用前缀表达式定义过程和引用过程构造前缀程序的思想和语法，掌握迭代与递归的应用场景和区别。

2. 能力目标

能够运用前缀表达式的基本运算、定义变量和定义过程构造前缀表达式程序，能够基于

C 语言运用迭代和递归构造函数。

3. 思政目标

树立质量意识,针对自己构造的迭代和递归函数,能够精心挑选典型的、苛刻的数据进行测试和验证。

3.2 计算系统与程序的关系

程序是实现计算系统复杂功能的一种重要手段,其本质是抽象、组合与构造,构造的基本手段是迭代和递归。递归是一种表达相似性对象及动作的重复的重要方式。各种编程语言仅仅是对程序思想表达规范化,以便计算机能够识别与执行。

1. 计算系统的设计与实现

首先,设计并实现系统可以执行的基本动作,这些基本动作可由现实物理世界的电子元器件稳定、可靠、低成本地实现,如 2.2 节所述的“与”“或”“非”以及“异或”四个基本动作。其次,算术运算加减乘除可由加减运算来实现,减法运算能转换为加法运算实现。最后,加法运算又可以转换为逻辑运算“与”“或”“非”以及“异或”四个基本动作来实现。

计算系统需要提供复杂的动作。复杂的动作不但千变万化,而且随使用目的的不同而变化。

2. 计算系统与程序的关系

程序由基本动作命令构造,它是若干指令的一个组合或一个执行序列,用以实现复杂的动作。例如,复杂动作组合表达式((A AND B) AND C) OR (NOT C),它被拆解为如下顺序执行的表达式。

① X=A AND B

② X=X AND C

③ Y=NOT C

④ X=X OR Y

上面 4 个顺序执行的表达式对应的计算系统指令为“与”“与”“非”“或”,4 个命令实际仅需三个基本动作“与”“或”“非”实现。指令是控制基本动作执行的命令。这 4 个顺序执行的表达式由程序执行机构自动解释其中的各种组合,并按照先后顺序调用指令予以执行。

简而言之,计算系统包含程序与执行机构两个方面。程序是一个复杂动作,是基本动作的各种方式的组合,其指令序列在程序执行机构的控制下分解为一个个基本动作由计算机执行。

3. 运用抽象和组合构造程序

人们基于算术运算和逻辑运算编写应用程序,更符合日常认知习惯。而“与”“或”“非”“异或”4 个基本动作和 AND、OR、NOT、XOR 4 个基本指令属于较低抽象层次,因而需要对较低抽象层次的动作和指令进一步组合后被命名成高层次的算术运算加减乘除。例如,

复杂动作(V1+V2)×(V3÷V4)÷V5,它被拆解为如下顺序执行的表达式。

① X=V3÷V4

② Y=V1+V2

③ X=X×Y

④ X=X÷V5

上面4个顺序执行的表达式对应的高层次计算系统指令为:÷、+、×、÷,这4个指令实际仅需三个高层次基本动作(即+、÷和×)实现。这4个顺序执行的表达式由程序执行机构自动解释其中的各种组合并转换成低层次的AND、OR、NOT、XOR指令,然后按照先后顺序调用指令予以执行。

3.3 基于前缀表示法的运算组合式程序构造

前缀表示法用运算符(即前述的指令)将两个数值组合起来,运算符在前面,将运算符表示的操作应用于后面的一组数值上,求出计算结果,通式为:(运算符 操作数1 操作数2)。其中,圆括号给出了组合式的边界,它是运算组合式的一部分,一组括号内只能有一个运算符,运算符是+、-、×、÷等,操作数可以有多个,运算符与操作数以及操作数与操作数之间用空格分开,示例如下。

一个运算符连加表示为(+100 205 300 400 51 304),一个运算符连减表示为(-500 205 50 100 10 20)。

1. 运算组合式的组合构造

可以将一个运算组合式作为参数代入另一个运算组合式中,通式如下。

①(运算符1 操作数1 操作数2)

②(运算符2 操作数a 操作数b)

③(运算符1(运算符2 操作数a 操作数b) 操作数2)

上面三个通式中,①中的操作数1被代入②后,得到了③,具体示例如例1所示。

例1 用前缀运算组合式组合构造算术表达式$\dfrac{149+\dfrac{15+3}{82-8\times 8}}{200-10\times 5}$。

解答:(/ 操作数1 操作数2)

(/(+149 操作数a2)(-200 操作数a4))

(/(+149(/操作数b1 操作数b2))(-200(×10 5)))

(/(+149(/(+15 3)(-82 操作数c1)))(-200(×10 5)))

(/(+149(/(+15 3)(-82(×8 8))))(-200(×10 5)))

前缀运算组合构造式的计算过程遵从四则运算规则:当一级运算(加减)和二级运算(乘除)同时出现在一个式子中时,它们的运算顺序是先乘除,后加减,如果有括号就先算括号内后算括号外,同一级运算顺序是从左到右。

例2 在例1的基础上,给出前缀运算组合构造式(/(+149(/(+15 3)(-82(×8 8))))(-200(×10 5)))的求值过程。

解答：(/(+149(/(+15 3)(-82(×8 8))))(-200(×10 5)))
(/(+149(/(+15 3)(-82 64)))(-200 50)))
(/(+149(/18 18))150))
(/(+149 1)150))
(/150 150))
1

2. 对运算组合式命名以简化运算组合式的构造

前缀运算组合式中的数据不便理解，而且长前缀运算组合式也不便编写。为便于理解和编写，可对计算对象进行命名，起到见名知意和拆解长前缀运算组合式的作用。

1) 命名计算对象

```
define 名字 操作数 2
```

define 为基本运算符，其作用为定义名字与操作数 2 关联，即操作数 2 用名字来表示，示例如下。

(define length 10)——定义 length 为 10。

(define width 5)——定义 width 为 5。

(define pi 3.1415926)——定义 pi 为 3.1415926。

(define radius 10)——定义 radius 为 10。

注意，不同类型的对象可以有不同的定义方法，这里统一用 define 来表示，在具体的程序设计语言中用不同的方法来定义。

2) 使用名字

(* length width)——使用 length 和 width 表达长方形的面积。

(* pi (* radius radius))——使用 pi 和 radius 表达圆的面积。

(define circumference (* 2 pi radius))——利用 pi 和 radius 定义周长的名字 circumference。

(* circumference 20)——对 circumference 乘以 20。

3) 执行计算

求值时用计算对象替换名字，直至具体数值时停止替换，开始运算，示例如下。

```
( * length width)
( * 10 5)
50
( * circumference 20)
( * ( * 2 pi radius) 20)
( * ( * 2 3.1415926 10) 20)
( * 62.831852 20)
1256.63704
```

3. 定义和使用新运算符构造组合式

1) 定义新运算符

将运算组合式 P，定义为以新运算符和形式参数引导的新运算符组合式，格式如下。

```
(define (新运算符 形参1 形参2 … 形参n) (含有形参1到形参n的运算组合式 P))
```

示例：

```
(define (square x) (* x x))
```

上例定义名字 square 为新的运算，x 为形式参数，当运用新运算符进行计算时以实际值代替 x，最后执行过程或函数体，此处过程仅为 x * x。

2）使用新运算符

新运算符的使用，类似于程序设计语言的函数调用，调用格式：(新运算符 实参1 实参2 … 实参n)，示例如下。

```
(square 10)
(square (+ 2 8))
(square (square 3))
(define (SumOfSquare x y)(+ (square x) (square y)))
(SumOfSquare 3 4)
```

3）执行新运算符

执行新运算符时以过程替换新运算符，直至具体数值时停止替换，开始执行计算，示例如下。

```
(SumOfSquare 3 4)
(+ (square 3) (square 4)
(+ ( * 3 3) ( * 4 4))
(+ 9 16)
25
```

3.4 迭代与递归

递归常被用来描述以自相似方法重复事物的过程，在数学和计算机科学中，指在函数定义中使用函数自身的方法，即 A 函数调用 A 函数。它把一个大型的复杂的问题转换为一个与原问题相似的规模较小的问题来解决，在每次调用自身时会减少一次任务量。递归是一个树结构，从字面可以其理解为重复“递推”和“回归”的过程，当“递推”到达底部时就会开始“回归”，其过程相当于树的深度优先遍历。计算理论证明递归的作用可以完全取代循环，因此在很多函数编程语言中习惯用递归来实现循环。递归能够解决的问题如下。

（1）问题是用递归定义，如 Fibonacci 函数。

（2）问题是用递归算法求解，如 Hanoi 问题。

（3）数据结构是递归形式，如二叉树、广义表等。

迭代是重复反馈过程的活动，每一次迭代的结果会作为下一次迭代的初始值，每迭代一次都将剩余的任务减少，即 A 函数调用 B 函数。

递归是迭代的一个特例，从理论上讲，任何递归都可以转换成迭代。递归中一定有迭代，但是迭代中不一定有递归，大部分情况下可以相互转换。在实际应用中，能用迭代的不用递归，因为递归有函数调用的开销，而且递归太深会带来堆栈溢出，递归比迭代效率要低。

递归性能不如迭代,但递归思路简单清晰,并且有时必须用递归求解,而迭代无法解决问题。例如,在实际开发过程中,有一张描述了实体之间的层次关系的表,要遍历所有实体之间存在的层次关系,即 $n:m$ 的关系,且事先不知道每个实体间的数量,用迭代无法实现,必须借助递归进行深层次递归求解。

下面以 Fibonacci 函数为例,说明递归与迭代的执行过程。

$$F(n)=\begin{cases}1, & n=0\\ 1, & n=1\\ F(n-1)+F(n-2), & n>0\end{cases}$$

假设 $n=5$,迭代计算过程如下。

$$F(0)=1$$
$$F(1)=1$$
$$F(2)=F(0)+F(1)=2$$
$$F(3)=F(2)+F(1)=3$$
$$F(4)=F(3)+F(2)=5$$
$$F(5)=F(4)+F(3)=8$$

同样地,对于 $n=5$,递归则分为递推与回归两个阶段,递归函数调用过程如图 3-2 所示。

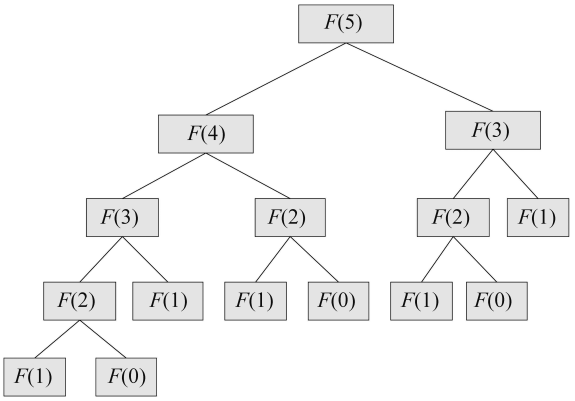

图 3-2 Fibonacci 函数递归调用树

图 3-2 是一个递归函数调用树,执行顺序为从左向右、自顶向下地递推,遇到叶子停止递推,开始自底向上地回归。

3.5 导产导研

3.5.1 技术能力题

(1) 给出 29×(17−5)−35/7 的前缀表达式表示与求值过程。

(2) 先用前缀表示法定义一个实现求解梯形面积 $\frac{(a+b)\times n}{2}$ 的前缀表达式,并利用该表达式求解 $a=4$、$b=6$、$h=5$ 梯形的面积。

(3) 假定函数 int Sum(int n)用于计算 $\sum_{i=1}^{n} i$，在具有健壮性的前提下，分别设计递推(迭代)和递归函数予以求和。

3.5.2 拓展研究题——美丽的分形

分形是指一个图像具有无穷自相似性。在自然和人工现象中，自相似性指整体的结构被反映在其中的每一部分中，如图 3-3 和图 3-4 所示的 Koch 雪花曲线和 Sierpinski 三角形就具有自相似性。

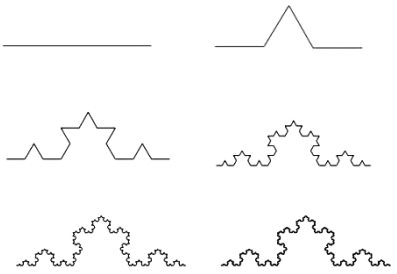

图 3-3 Koch 雪花曲线

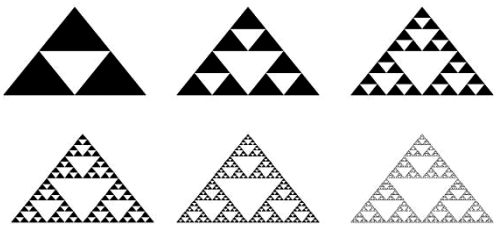

图 3-4 Sierpinski 三角形

1. Koch 雪花曲线

如图 3-3 所示，设 E_0 为单位直线段，三等分后，中间一段用与其组成等边三角形的另两边代替，得到 E_1，对 E_1 的 4 条线段的每一条重复以上做法，得到 E_2，以此方法重复，可得 E_n。当 n 趋于无穷，得到的极限曲线就是 Koch 雪花曲线。

2. Sierpinski 三角形

如图 3-4 所示，Sierpinski 三角形是等边三角形四等分去中间小三角形所得极限图形。

请读者设计实验方案，使用可视化图形编程工具，如 MFC，运用递归绘制 Koch 雪花曲线和 Sierpinski 三角形，基于不同内存空间和不同频率的 CPU，观察、记录并分析递归深度和所需时间的关系。