1988年第1期　中山大学学报 ACTA SCIENTIARUM NATURALIUM UNIVERSITATIS SUNYATSENI　No. 1, 1988

一个基于模块PASCAL的结构编程环境

李师贤　雷晖
（计算机科学系）

摘　要

本文首先介绍在PASCAL语言中如何引入模块功能，使其更利于数据表示的抽象，然后着重讨论一个基于这种模块PASCAL的编程环境，用以支持自顶向下的结构化程序设计方法，该系统主要包括两个功能块：模块PASCAL的语法制导编辑和预处理。

关键词　编程环境，逐步求精，模块，PASCAL，语法制导编辑器

程序编制的传统方法是使用标准工具链，这种方法的编辑程序不具有语言知识，如果到工具链执行的后期发现了程序的问题，需再返回到编辑程序；这种方法也不能支持较高抽象级别上的程序设计和某种程序设计方法。为了提高软件生产率和减少程序设计工作的复杂性，必须改善现有的编程方式。本文介绍一个基于模块PASCAL的结构编程环境的设计与实现。

1　在PASCAL中引入模块功能

为了控制和解决大型软件系统开发中的复杂性问题，人们提出利用软件工程的原理来分解系统，自顶向下的结构化设计方法能为抽象分解提供很好的一致性标准，这种逐步求精的程序设计方法从最能直接反映问题的体系结构的概念出发，设计出一个抽象算法（由抽象数据及其上的抽象操作构成），然后对抽象算法作进一步的求精而进入下一层的抽象，在解的较低层给出实现较高层动作的基本操作，在整个过程中，以信息隐藏为准则。

除了方法，我们还需要程序设计语言这样的形式工具来表达和执行设计。PASCAL无疑是很流行的，而且可以给算法提供层次较深的嵌套结构，但PASCAL是一个分程序结构的语言，它具有如下缺点：

①在源程序中，某一特定抽象数据类型的实现（相应的类型，变量，过程等的定义说明）被割裂开来，不得不分别出现在有关的说明部分，以致影响了程序的清晰性和可读性；

②任何标识符只有在它被说明的分程序中才是可用的，因此若某一抽象数据上有两个以上的操作时，该抽象数据的定义说明将极不自然地出现在包含这些操作的分程序中，而不是实现操作的过程的内部，这样数据实现细节对使用其上操作的分程序来说便不再

本文1987年9月收到

是隐藏的，甚至该分程序中所有过程都可以改变它的属性，这样就不能保证抽象数据类型的使用与实现的无关性；

③一个大型软件系统的开发维护往往是由若干个程序员合作完成，显然分程序结构并不能对这种工作方式提供有力的支持，同时也不利于源代码的重用。

简言之，PASCAL语言未能实现数据表示的逐层抽象，不利于自顶向下的结 构化程序设计。

为克服上述缺点，我们对标准的PASCAL作一些扩充，以引入模块功 能。模 块的划分基于SIMULA67中类（CLASS）的概念，把数据及其上的操作集中起来 作为一种抽象数据类型。图1是模块PASCAL的语法图。

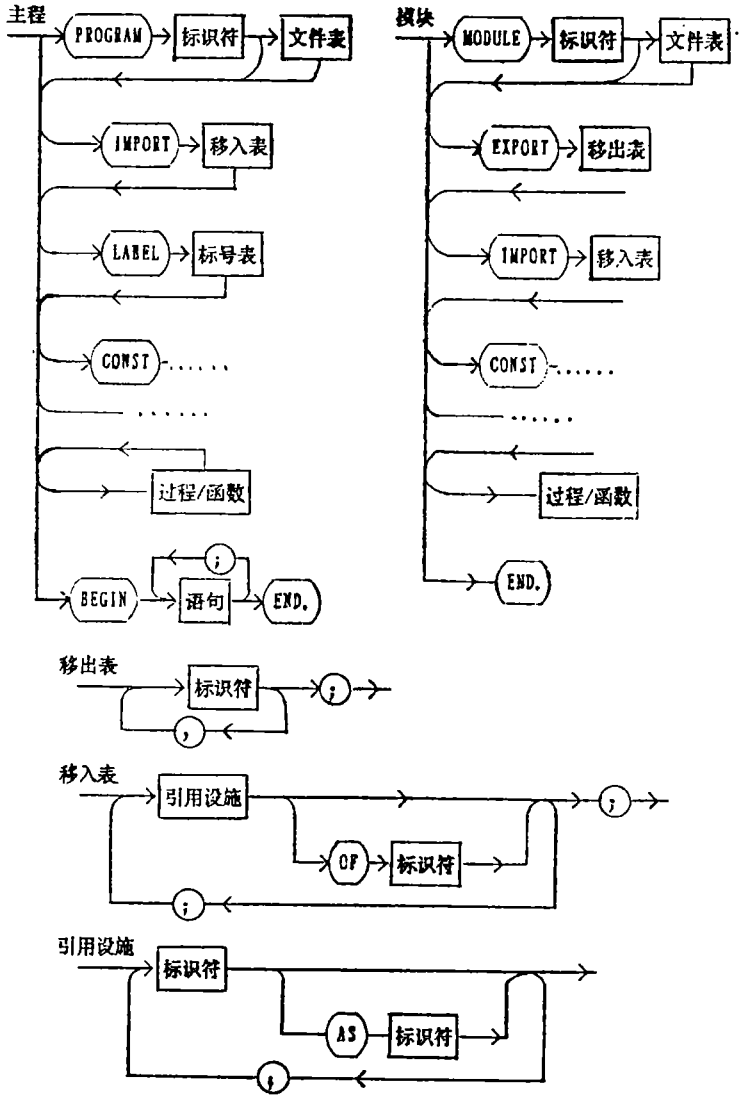

图1 模块PASCAL的语法图

Fig.1 Syntax graphs of modular Pascal

由图1可以看到，主程类似于传统的PASCAL程序，模块给出数据的内部结构和操作细节，而通过移出表提供给外部使用的只是掩藏了这些实现细节的抽象的数据和抽象的操作，主程和模块均可以使用其他模块定义的可移出对象，只要将这些对象列入移入表中；允许引用设施在不同的模块中具有不同的名字。

下面是用模块PASCAL书写的一个示例：

```
PROGRAM main;
IMPORT a, b AS b1, proc, proc1 OF
        lib;
CONST k=5;
TYPE t=RECORD
         c1: integer;
         c2: a
       END;
VAR b: t; c: b1;
PROCEDURE p;
   BEGIN… END;
BEGIN
   b. c1:=0;
   proc (b. c2);
   proc1 (c)
END.
MODULE lib;
EXPORT a, b, c, proc, proc1;
CONST h=3;
TYPE a=ARRAY [1. .h] OF real;
     b=ARRAY [1. .h] OF boolean;
VAR c: integer;
PROCEDURE proc (x: a);
   BEGIN … c:=1; … END;
PROCEDURE proc1 (v: b);
   BEGIN…END;
END.
```

模块PASCAL在物理上隐藏每一层不必要的细节，使设计决策局部化。

2　编程环境

当今的程序自动化领域，人们普遍致力于建立程序工具和环境，本文介绍的系统是一个基于前述模块PASCAL，支持自顶向下的结构化程序设计方法的编程环境。它包括两个工具：语法制导的编辑程序（结构编辑器）——提高逐步求精的程序设计自动化程度；模块PASCAL的预处理程序——将无语法错误（由语法制导的编辑程序保

证）的模块PASCAL源代码加工成一个通常的PASCAL程序，使其可被任何标准的PASCAL编译程序接受。

图2给出了用户观点的环境工作流程。

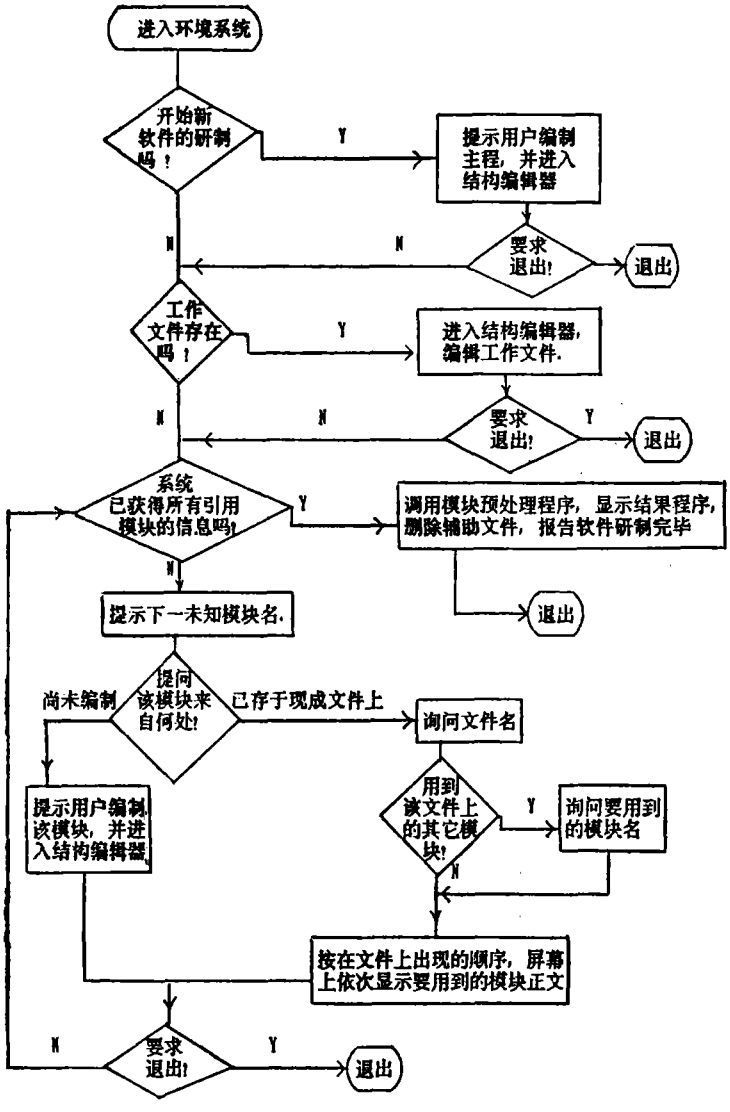

图2 用户观点的环境流程

Fig.2 Flowchart of software developing

在整个工作流程中，与用户打交道最多的是结构编辑器，这里主要介绍它的用户界面。

系统提倡的输入方式是：自顶向下的结构展开和由底向上的表达式文本输入相结合，允许用户按文本方式输入任一语法单位。

结构编辑器设置了三个窗口：结构编辑窗口SE（用作以模式展开方式编程），文

本方式编辑窗口TE和标识符说明窗口SD。

首先来看一下主程的编制方式。

开始，屏幕被分为上下两块，分别充当SE和TE. SE显示如图 3 画面。

```
PROGRAM    〈ID〉;
/DECLARATIONS/
BEGIN
 {[S]TATEMENTS}
END.
```

图 3　SE初启画面("〔〕"代表光标位置)

Fig.3　Initial picture of SE window

图中的"/DECLARATIONS/"只是说明在结果程序中，标识符定义部分的位置，它在SE中并不能真正被展开成标识符说明（标识符的说明必须进入SD，参见后面所述）。除此以外，SE显示三种不同的符号：

① 必展符，由"〈"和"〉"界定，必须按其相应的提示内容加以展开；

② 选展符，由"{"和"}"界定，它可以被扩展，否则，表明对应的内容不需要，在最后将被系统自动删除；

③ 标准符，即源程序中的终结符，当一个完整的主程或模块编制完毕，只存在有标准符。

为展开必展符和选展符，用户可将光标移至界定对应符的括号内，并敲入缩略形式表示的某种构造即可；若该符属于表达式一类，系统会自动将光标移至TE，供用户按文本编辑形式输入，若输入不符合语法，系统将拒绝接受，并给出警告，光标返回原来位置，而所输入的内容保留在TE的缓冲区，可供进一步的编辑。

例如，光标位置如图 3 所示，若用户敲进"W"（意味该语句将被扩展成WHILE语句），则屏幕显示如图 4 所示。

```
                                  ** SE **
PROGRAM
/DECLARATIONS/
BEGIN
    WHILE 〈CONDITION〉 DO〈STMT〉;
    {STATEMENTS}
END.
- - - - - - - - - - - - - - - - - - - - - - - - -
condition:                        ** TE **
[ ]
```

图 4　屏幕显示例图

Fig.4　Instance of screen display

"〈STATEMENTS〉"被展开成WHILE语句后，光标移至"〈CONDITION〉"，它属于表达式，故系统会自动将光标移入TE，要求用户输入条件表达式来展开"〈CONDITION〉"，原来"〈CONDITION〉"的位置将以加亮方式显示。

在SE内，用户命令还包括光标移动及以语法结构为单位的增删操作。

当语句部分编制完毕，系统要求用户在TE内按文本方式输入有关入口信息，然后，屏幕切换为SD窗口，让用户进行标识符的定义说明，系统已从语句部分获得有关标识符的大量有用信息，例如

IF b THEN a [i DIV 2]:=p^

系统便可根据上下文信息推断b为布尔型，a为一维数组，i为整型或子界类型，p为指针等。这样，在SD内，系统逐屏显示它推断可能属于同一类的标识符，并给出菜单提示，用户只要将光标移至要说明的标识符，并选择恰当的菜单序号即可，关于语句标号定义和出现在首部的文件表，系统会自动补足。

最后，系统根据识别出的过程/函数标识符，要求用户逐层细化。

考虑到用户个人的习惯，系统允许将某一部分结构移至TE内进行文本方式的编辑，也允许用户随时在TE内进行标识符说明。

系统提供了一种非常灵活的屏幕显示方式，即所谓漂亮格式打印(PRETTY PRINTING)。用户可要求系统按通常方式显示；或只要求显示一个语法结构单位，系统通过省略显示其中某些次要的部分，使整个结构轮廓在一个画面上显示出来，省略显示的例子如：

BEGIN…END;

IF a=b THEN b:=2 ELSE…;

至于模块的编制，则首先进入TE窗口，编辑器要求说明出口信息，这相当于以规定抽象数据及其上操作的方式给出模块的规格说明；接下来定义入口信息和说明标识符，然后系统要求用户逐个编制过程/函数。

在本环境中，程序的编制过程是由主程到过程/函数，再逐层深入到提供数据抽象的各模块，从而遵循了由顶向下逐步求精的结构化程序设计原则。

3 实现技术

关于语法制导编辑程序，按照通常的作法，以内部树表示语言的结构，在本系统中，内部树基于"左儿右兄弟"方法实现为二叉树形式。为内部树设计一组结点构造和树周游操作，用户的外部命令被解释成一串内部树上的操作。

SE窗口的工作是根据内部树中结点的不同状态和用户敲入的不同命令进行相应的动作。当用户命令为光标移动，状态转换或结点增删时，动作不依赖于结点状态；只有对应于语句的结点才能接受模式展开命令。

TE窗口主要对已编辑的代码进行语法分析：对表达式以上的非终结符用递归下降算法，相应动作为构造内部树的一个子树；对表达式则以算符优先算法，从表达式中捕捉有关符号的尽可能多的信息，并填符号表，具体说明如下：

①算符优先算法所分析的是一个操作数和操作符的串，这就要求扩展词法分析器，使其输出不是简单的单词符号，而是操作数或操作符，对于属于构造类型的标识符，我

们可以通过在词法分析器中辨别其后缀而加以分类。

②对于属于简单类型的标识符，可以在归约时通过区分对其施加的运算符而分类。

③判定语句中出现的标识符是常量还是变量，可看它的第一次出现是否在表达式中。

SD窗口根据符号表的内容来提问各标识符。

为实现漂亮格式打印功能，需要为每一种子树构造一个显示模式。

一棵子树对应于一个语法结构，生成子树的产生式可归为两类：聚合型和列表型，下面是两个例子：

聚合型：statement⟶ IF expression THEN statement ELSE statement

列表型：statement⟶ BEGIN statement {; statement} END

对于聚合型的产生式，显示模式由一串广义的操作符（即标准符）和操作数（对应于产生式中的非终结符）组成。表示上述IF语句的子树，其显示模式为

IF $1 THEN $2 ELSE $3 （$i表示显示根的第i个儿子）。

对于列表型产生式，显示模式包括表头，表尾和界符，在表示复合语句的子树的显示模式中，它们分别是“BEGIN”，“END”和“;↑”（↑代表换行）。

我们考虑过程SHOW(A, W),它将内部树中以A为根的子树在区域W显示出来。

SHOW(A, W)的执行可以分成下面三步：

① MEASURE(A). 暂时撤开W的空间限制，以后序遍历方式计算显示子树每个结点所需区域的大小，最后得出显示以A为根的子树所需区域的大小；

② FIT(A, T). T为一个二维字符型数组，它与区域W的“面积”相同，FIT过程通过递归调整，以填写T的方式，确定以A为根的子树的显示方案；

③ DISPLAY(T,W). 将T的内容输出到屏幕的W区域。

模块预处理程序的基本思想是：按照各模块之间相互引用的关系，利用先深搜索，确定它们的一个拓朴次序，使被引用的模块在前面；然后按照次序，将模块的常量，类型，变量等说明分别归并到主程的相应部分，形成结果程序的说明；对各子过程/函数作向前引用说明FORWARD.最后，将各模块定义的过程/函数抄入结果程序，再抄入主程的过程/函数以及主程的语句部分。在后面两步，必须使不同名的引用设施具有相同的名字，并利用一个随机数产生器解决全局标识符定义冲突问题。

我们在前面曾给出了用模块PASCAL书写的一个主程和一个模块，用预处理程序将其合并的结果如下（其中主程的引用设施b1被还原成原来的名字b，主程中的b与c由于名字冲突而被换为b7与c5）。

```
PROGRAM main;
CONST h=3;
      k=5;
TYPE a=ARRAY [1..h] OF real;
     b=ARRAY [1..h] OF boolean;
     t=RECORD
       c1: integer;
```

```
        c2: a
      END;
VAR c: integer;
    b7: t;    c5: b;
PROCEDURE proc (x: a); FORWARD;
PROCEDURE proc1 (v: b); FORWARD;
PROCEDURE p;
  BEGIN…END;
PROCEDURE proc;
  BEGIN … c:=1; … END;
PROCEDURE proc1;
  BEGIN … END;
BEGIN
  b7. c1:=0;
  proc (b7. c2);
  proc1 (c5)
END.
```

4 结束语

我们用极少的语言扩充引入模块功能，并提出一种改进的程序开发系统，有利于用户将程序分解成模块，得到可重复使用的软件，提高软件生产率，增强软件可靠性。

本文引进的模块PASCAL，只考虑一种特殊的模块性：允许一个方向上的显示信息交换，从而避免模块之间的循环依赖性，保证模块之间的逻辑依赖性为半序关系。

由于结构编辑器必须与预处理程序及标准PASCAL编译程序链接，因此本环境不提供增值式的程序构造和源语言级的语言调试功能。

环境稍进一步的拓展可以从以下两方面入手：

主程改造：提供工具允许用户将使用过的主程改造成模块，该功能可以按类似结构编辑器的方式实现；

模块库生成：提供操作系统级的文件处理，方便用户将若干模块并置成一个模块库。

参 考 文 献

[1] M. V. Zelkowitz, A Small Contribution to Editing with a Syntax-irected Editor, *ACM Software Engineering Notes*, 9(1984), 3.

[2] C. N. Fischer, Anil Pal and Daniel L. Stock, The POE language-based Editor Project, *ACM Software Engineering Notes*, 9(1984), 3.

[3] Teitelbaum, Tim and Thomas Reps, The Cornell Program Synthesizer: a syntax-directed programming environment, *Comm. ACM*, 24(1981), 9.

〔4〕L. Allison, Syntax-directed Program Editing, *Software-Practice and Experience*, 13(1983) 5.

〔5〕B. Meyer, J. M. Nerson and S.H.Ko, Showing Programs on a Screen, *Science of Computer Programming*, 1985, 5.

〔6〕M. Ancona, L. D. Floriani, G. Donero and S. Mancosu, Integrating Library Modules into Pascal Programs, *Software-Practice and Experience*, 14(1984) 5.

〔7〕王振宇等，结构编辑程序的综述与分析，计算机科学，1986,6.

〔8〕仲萃豪等，程序设计方法学，北京科学技术出版社，1985

A Structured Programming Environment for Modular Pascal

Li Shixian　　*Lei Hui*

Abstract

We start with an introduction of modularity into programming Language Pascal. We then describe in detail a structured pregramming environment for that modular Pascal. The environment consists of a syntax-directed editor and a preprocessor, and is meant for programming by stepwise refinement. The user interfaces are described and the technical issues presented

Keywords programming environment, stepwise refinement, modularity, Pascal, syntax-directed editor