北海道大学工学部研究報告
第108号（昭和57年）

Bulletin of the Faculty of Engineering,
Hokkaido University, No. 108 (1982)

Pascal プログラミングのための支援環境システム

宮本衛市　竹村伸一

（昭和56年12月26日受理）

A Supporting Environment System for Pascal Programming

Eiichi MIYAMOTO and Shin-ichi TAKEMURA

(Received December 26, 1981)

Abstract

This paper describes a Pascal programming system developed to support both debugging and editing of Pascal programs conversationally and structurally. As the system simulates the facilities as a Pascal machine for the programmer to run and edit his program, it accepts all of its commands on the source level of the program. Under the support of the system, the programmer can run his program at will, monitor its behavior including the dynamic data structure, and edit it without the management of the operation mode which the system is subject to.

The main features of the system are as follows:

(1) It manages machine code modules directly to execute programs efficiently and in a variety of execution modes.

(2) It can print out the updated values of variables concurrently with running, and the entire structure of variables including dynamic structure at break.

(3) It can restore the status of a program according to its structure.

(4) It accepts program corrections whenever the programmer detects some bugs, and continues to execute the updated program.

1. はじめに

計算機に占めるソフトウェアのハードウェアに対する比重は，最近ますます大きくなり，集積回路化の技術の目覚しい発達によるハードウェアの低廉化と，計算機の大型化によるソフトウェアの大規模化の相乗効果がこれに拍車をかけている。そのため，ソフトウェアの生産性の向上は緊急の課題であるが，ソフトウェアは人間の高度な知的活動の産物であり，ソフトウェア技術の進歩は，人間の知的活動の高能率化に強く依存している。これまでにも，数多くのプログラミング言語やプログラミングの方法論が提案され，人間がより高度なレベルでプログラムを作成するための言語や，プログラミングに対する種々の規範が与えられてきた[1]。しかし，プログラミングをするのが人間である以上，プログラムに誤りの混入は避けられず，虫取りあるいはテストに多大な労力を費しているのが現状である。

一方，計算機システムの向上，なかでも TSS の普及により，人間と計算機との間の情報の授

情報工学専攻　情報システム工学講座

受が秒単位で行なえるようになり，人間の知的活動に計算機が貢献できる能力が一段と飛躍した。すなわち，計算機は単にプログラムを翻訳し，実行してその結果を渡すだけでなく，人間の知的活動を支援するための環境を提供することができるようになったのである。もちろん，コンパイラも一種のソフトウェア・ツールと考えることもでき，単に言語翻訳をするだけでなく，プログラム診断機能，虫取り機能，実行解析機能等を備えてシステム化したものが開発されている[2~4]。しかし，それらの機能を行使するために，あらかじめプログラムの中に何らかの指示を挿入しておくのでは，即応性や融通性に欠ける。事態に即応した臨機応変の対応が計算機との間で行なえることが極めて望ましい。TSS は人間と計算機を実時間で結合させるものであり，支援環境としての基礎となるものである。最近，TSS のもとで人間のプログラミング活動を総合的に支援する環境システムが数多く報告されている[5~14]。これらのねらいはそれぞれ異なってはいるが，いずれも TSS 環境下で，プログラムの作成支援，実行制御，入出力動作，診断解析等を会話的に行おうとするものである。

本論文は，Pascal プログラミングを強力に支援するために開発したシステム PENSES (Programming ENvironment SupErvisory System) について述べたものである。一般に，Pascal で書かれたプログラムは静的な構造を有しており，PENSES はこの構造に基づいたプログラムの実行制御および編集処理を，プログラムの指示により会話的に行なう。プログラマの指示はすべてプログラムのソースレベルで行なうので，プログラマはその指令が実行制御であるのか，または編集処理であるのかを全く意識する必要がない。あたかも，Pascal のソース・プログラムがそのまま実行される Pascal マシーンを駆動していると考えることができる。PENSES のもとで，プログラマは作成途上のプログラムを意のままに実行させながら，その挙動を観察し，意に反した振舞いからその原因を追求し，即座に修正する一連の作業を，TSS 端末を通して実時間で行なうことができる。本システムは，筆者らが先に開発した構造的エディタ POESY[6] と，構造的実行制御システム INSIDER[9] を結合し，より総合的な支援環境システムとしたものである。

2. システムの設定する環境と支援機能

PENSES は Pascal マシーンをソフトウェア上でシミュレートしている。したがって，プログラマは Pascal プログラムがそのまま実行されるものと考えればよいのであって，各手続きの実行本体が機械に対する命令に，変数がデータ領域の番地に相当する。しかし，実行本体は入れ子構造をもつ文で記述されていて，機械語のように，命令ごとに切り離して考えることができない。そこでプログラムを節を単位とした木構造で表現し，節を制御対象とすることにより，入れ子構造をもつ文の切り離しが可能になり，かつ文を代表するものと考えることができる。一方，変数は有効範囲をもっているので，変数名のみでは一義的に決まらず，どこを基準とする変数であるかを明示する必要がある。なお，この Pascal マシーンでの演算は式の評価そのものであり，プログラム・カウンタに対応するのが現行節，すなわち

```
program pascal(input,output);
  type ptr=↑rec;
       rec=record key:integer; next:ptr end;
  var p,q:ptr; k:integer;
begin p:=nil;
  while not eof do
    begin new(q);
      with q↑ do
        begin read(key); next:=p end;
      p:=q
    end
end.
```

図1 Pascal プログラムの例

次に実行すべき節を指している節指示子である。

2.1 階層的木構造によるプログラムの表現

Pascal プログラムは，基本的には並置構造と入れ子構造を骨格として構成される。前者は識別子の定義・宣言，複合文における文の列挙などで，それぞれ完結した記述が並記される。それに対し，後者はレコード内のフィールド記述や配列の要素記述，あるいは局所的な手続き・関数宣言，構造文が内部に抱える文の記述などのように，再帰的な構文規則に基づいて，同じ構造を持ちうる部分構造が入れ子として記述される。この両構造を把握するために， PENSES は Pascal

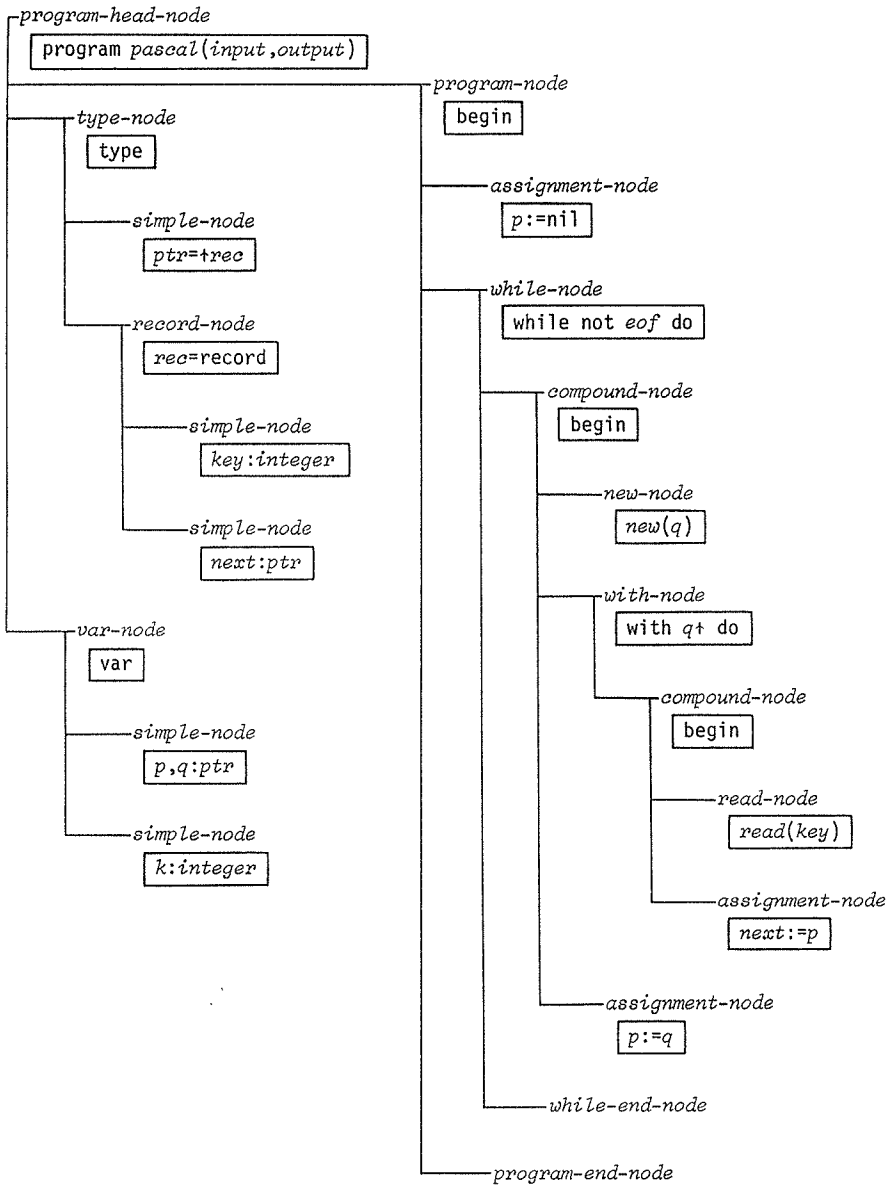
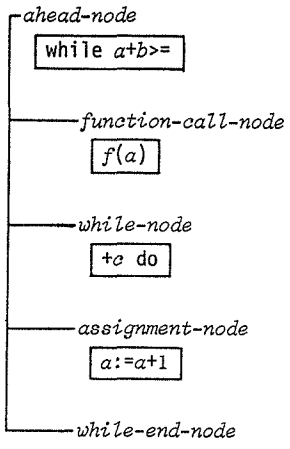


図2 図1に示すプログラムの木構造

プログラムを多分岐の木構造に変換する。ここで，木構造の根はプログラム全体を指す節であり，各節はプログラム・手続き・関数，定義・宣言部，識別子の定義・宣言，実行本体，または文を指している。木構造上の多分岐が上記の並置構造に対応し，節とそれが指す部分木との関係が入れ子構造に対応する。内部構造を有しない記述，たとえば単純型をもつ型名定義または変数宣言，関数呼び出しを含まない代入文や手続き呼び出し文などの節は木構造上の端点を構成する。図2は，図1に示すプログラム例を木構造に展開したものであり，各節には対応するプログラム・テキストを示している。図2で，たとえば **while** 文は **while** 節を根とする部分木を構成しており，複合節と **while** 終端節へと枝分れしている。ここで，**while** 終端節は対応するプログラム・テキストを持っていないが，2回目以降の繰り返しを担当するために設けた節である。同様な理由で，**for** 文に対しても **for** 終端節を設けている。

木構造上，手続き呼び出し文を端点で表わすが，実行時には呼び出す手続きを部分木として動的に指す節と考える。一方，関数呼び出しを含む式は，関数呼び出し前の処理，関数呼び出し処理および復帰後の処理に分割して考える。その例を図3に示すが，実引数中に再び関数呼び出しがあるときや，復帰後に再び関数呼び出しがあるときには，対象となる節を再び分割する。




```
while  a+b>=f(a)+c do a:=a+1
```

図3 関数呼び出しを含む木構造

PENSES のもとでプログラムを実行することは，見かけ上，実行本体のある部分木の各節を，節の種別に応じて実行していくことであり，プログラムの修正は基本的には部分木の修正である。節を指示することによって，その節を根とする部分木を指すこともできるが，指示された節がその節のみを指すのか，または部分木を指すのかはコマンドにより決まる。節の指示は，宣言・定義部では識別子名により，実行本体では，〈行番号〉・〈節の出現順位〉により行なう。

2.2 会話形式によるプログラミング支援

PENSES はプログラマからのコマンドによる指令で動作する。表1に PENSES の有するコマンドを一覧表にして示す。同表にも示すように，便宜上コマンドを5つに分類する。

(1) 指示コマンド　これらは PENSES 自身に対する指示である。なお，PENSES は内蔵するコンパイラにコマンドの字句解析も委ねているので，コマンドの中に注釈を含めることがで

表1 コマンドの機能の概要

分類	コマンド名	省略名	機能
指示コマンド	source	SO	プログラムの翻訳，節展開等の準備
	bye	BYE	終了指示，制御をTSSに戻す
	jump	J	現行節の変更
設定コマンド	break	B	指定した節に中断条件を設定
	break cancel	BC	指定した節から中断条件を削除
	display	D	変数の動的表示を指示
	display cancel	DC	変数の動的表示を解除
	restoration procedure	RSP	手続きごとに復元することの指定
	restoration node	RSN	節ごとに復元することの指定
	restoration cancel	RSC	復元指定の解除
実行コマンド	run	R	プログラムの始めから終りまで実行
	continue	C	現行節から終りまで実行
	step	S	1つの文の実行
	advance	A	1つの節の実行
	execute	E	オペランドで指示した文の実行
	restoration	RS	最も近くの復元可能な節まで戻る
診断コマンド	list	L	変数のデータ構造に応じたダンプ出力
	frequency	F	節の実行回数の表示
	where	W	節に対応したソーステキストの表示
	trace nodes	TN	実行してきた節の履歴の表示
	trace procedures	TP	呼び出された手続きの履歴の表示
編集コマンド	string	STR	文字列としての編集
	label	LA	名札の編集
	const	CO	定数の編集
	type	TY	型の編集
	var	VA	変数の編集
	statement replacement	SR	文の置換
	statement insertion forward	SIF	文の前方への挿入
	statement insertion backward	SIB	文の後方への挿入
	statement appendant	SA	分岐文の条件節の追加
	procedure/function deletion	PFD	手続き・関数の削除
	procedure/function replacement	PFR	手続き・関数の置換
	procedure/function insertion	PFI	手続き・関数の挿入
	pretty printing	PP	テキストの清書

き，注釈を通してコンパイラ・オプションを指定することができる。jump コマンドは execute コマンドによる **goto** 文の実行と同じ働きをするが，名札の代りに節を指定して飛ぶことができる。ただし，現行節のある手続き*から飛び出すことや，他の構造文の中へ飛び込むことはできない。

次に例を示す。

(＊$L—＊) SO ……ソーステキストを出力しないでプログラムを翻訳し，木構造へ展開する。
J 40.1 ……40.1で示される節へ現行節を移す。
J ……節位置の指定がないときには現行節をプログラム節へ移す。

(2) 設定コマンド　これらは実行コマンドの指令に先立って，中断条件，動的表示または復元指示をあらかじめ PENSES に伝えておくものである。中断条件は Pascal の構文規則に従う論理式で記述し，中断させたい節に割り当てておく。割り当てられた節では，その節の実行後中断条件を評価し，真であれば次に実行すべき節を現行節として実行を中断する。動的表示は指定した変数が代入文で更新されるたびに，その新しい値を表示するもので，特定の代入文のみを対象にする場合と，位置に関係なく更新されたら表示する場合とがある。

PENSES はプログラムを復元する機能を有している。これはプログラムの状態を過去にさかのぼって復元していくものであり，木構造の根の方に向って，各節が実行に入る直前の状態に復元する。そのためには，置き換えられる以前の変数の値を各節に待避させておかなければならず，実行時間と記憶容量を大量に必要とするので，復元の舞台とする手続きを指定しておく。restoration node コマンドは木構造の節に沿って復元できるようにすることへの指示であるのに対し，restoration procedure コマンドは手続きへ入ってきた直後の状態へ復元できるように指示するものである。次にいくつかの例を示す。

B 20 *true*; ……20.0の節を実行後，無条件で中断させる。
B 30 *i*>＝20; ……30.0の節を実行後，*i* が20以上であれば中断させる。
BC 20……20.0の節に設定した中断条件を解除する。
D 40……40.0の代入文を実行後，新しい値を表示する。
D ＊ *a*, *b* ……現行節から見える変数 *a* および *b* が更新されるたびに表示する。
RSN *search* ……手続き *search* に節単位で復元できるように指示する。

(3) 実行コマンド　これらはプログラムの多彩な実行を PENSES に指定するものである。run コマンドはプログラム節からプログラム終端節までの実行を指示するのに対し，continue コマンドは節指示子の指す現行節からプログラム終端節までの実行を指示する。step コマンドは現行節を根とする部分木，すなわち文の実行を指示する。ただし，その文が手続き呼び出し文であったり，関数呼び出しを含む式をもっているような場合には，呼び出された手続きまたは関数の実行をすべて含む。すなわち，step コマンドの実行後は，見かけ上次の文の先頭で停止するので，再帰的呼び出しを行なっても，再帰的な呼び出しの深さを変えないで，同一の深さの手続きに戻って停止する。step コマンドの処理中に，その処理範囲を逸脱するような**goto** 文があった場合には，そこで実行を中断する。また，上記の実行コマンドを実行中に，設定されていた中断条件が成立したり，コマンドで指定した打切り時間を超過したり，端末から割り込みを受け付けたりしたときには実行を中断する。

advance コマンドは現行節のみの実行を指示する。step コマンドがプログラムの並置構造に沿っての実行指示であったのに対し，advance コマンドは入れ子構造に沿った実行指示をするこ

＊以後，手続き，関数および主プログラムを総称して手続きと呼ぶことにする。

とに対応する。execute コマンドは Pascal の構文規則に従う文を通して，プログラムの状態を監視したり，さらには変更したりするためのもので，入力された文はその場で直ちに実行される。なお，上述した実行コマンドを実行中に異常な事態が検出されると，検出された時点の節の直前で異常の種別を通知して中断する。そのため，**nil** をもつポインタ変数で参照したときや，四則演算時の異常，添字範囲の逸脱等が，見かけ上発生直前の状態で検出される。

restoration コマンドは現行節から最も近くにある復元可能な地点へ復元する。すなわち，現行節のある手続きが restoration node コマンドで指定されていた場合には，木構造上で長兄側の節か，さもなければ親の節へ復元する。一方，手続きが restoration procedure コマンドで指定されていると，手続きを実行開始する直前の状態へ復元する。ただし，名札が定義されている文から復元するときには，**goto** 文による合流が考えられるので，親の節へ一足飛びに復元する。もし，現行節のある手続きに復元指定がなされていないと，呼び出した手続きをさかのぼって復元可能な節を探す。次に，実行コマンドの例をいくつか示す。

R 20 ……プログラムを始めから実行するが，20 ms 経つと中断する。
S ……現行節を根とする文を実行する。
A ……現行節のみを実行する。
E *writeln* (*s*, *t*); ……変数 *s* および *t* の値を出力する。
R S ……最も近くの復元可能点へプログラムを復元する。

(4) 診断コマンド　list コマンドは一種のメモリダンプを行なうものであるが，単に一定区域のメモリ内容を出力するのではなく，指定した変数のデータ構造全体に渡って，データの種別に応じたダンプ出力である。したがって，構造型の変数であればすべての要素を，またポインタ変数であれば動的変数で構築される動的なデータ構造全体を出力する。step および advance コマンドが構造に即した実行制御であるのに対し，list コマンドは構造に即した監視機能である。次に診断コマンドの例をいくつかあげよう。

L *tblarray*; ……配列変数 *tblarray* を出力する。
L *pointer*;……ポインタ変数 *pointer* を源とする動的データ構造を出力する。
F *sub* ……手続き *sub* の各節の実行回数を表示する。
T N 50 ……実行してきた節の履歴を現在からさかのぼって50個表示する。
T P 20 ……呼び出された手続きの履歴を現在からさかのぼって20個表示する。

(5) 編集コマンド　これらのコマンドは Pascal プログラムを識別子の定義・宣言の全体またはその要素，文，および手続きを対象として編集するが，文字列として編集することもできる。定義・宣言部および文の編集は現行節のある手続きを対象とする。label, const, type および var コマンドで指定した名札または識別子がプログラム上に存在していなければ挿入し，存在していれば置き換え，また識別子のみを指定すればプログラム上から削除する。構造型の要素を編集するために，レコード型では‘・’に続けてフィールド名を指定し，他の構造型では‘＊’を付けると，その要素の指定を意味する。さらに配列型の場合，‘／’を付けると添字の編集を意味する。次に定義・宣言部の編集コマンドをいくつか示す。

L A 30, 40 ……名札の宣言を追加する。
C O *pai* = 3.1415927; ……定数名 *pai* を定義する。
T Y *rec*. *field* : *integer*; ……レコード型 *rec* にフィールドを追加または置換する。
T Y *matrix* ＊ ＊ : *real*; ……2次元配列変数 *matrix* の要素の型を *real* にする。
V A *vector* ／ : 1 ‥30; ……配列変数 *vector* の行添字を変更する。
V A *del* ……変数 *del* を削除する。

文の編集では対象となる場所を節で指定する。また，既存の文を取り込んでプログラムを改造するようなときには，節を‘<’と‘>’で囲んで文の列として組み入れることができる。以下に例で示す。

```
SR   30  if a>0 then <30>;  ……30.0の文を if 文に改造する。
SIF  40  p:=nil;  ……40.0の文の前に代入文を挿入する。
SIB  50  <100>;  ……100.0の文を50.0の文の後に移動する。
SA   60  a:=a-b;  ……60.0の if 文に else 節を追加する。
SA   70  '9':n:=n+9;  ……70.0の case 文に条件節を追加する。
```

手続きの編集には削除，置換および挿入があるが，その位置を既存の手続き名で示す必要がある。対象となる手続き名のみではあいまいになる場合には，それを含む親の手続き名を先行させる。以下に例で示す。

```
PFD  operation. add  ……手続き operation の中にある関数 add を削除する。
PFR  sub  ……手続き sub を以下のように変更する。
   procedure sub …… ;
PFI  search  ……手続き search 内の先頭の手続きとして以下を挿入する。
   procedure try; begin <100..200> end;
PFI  search (find)  ……手続き search 内の手続き find の後に以下を挿入する。
   procedure try; begin <100..200> end;
```

上述の構文的な編集ではむしろ繁雑なときのため，節に割り当ててあるソーステキストを文字列として編集するのが string コマンドである。

```
STR  20  "symbol"sign"  ……20.0の節の文字列 string を sign に置き換える。
STR  30  "100"200"*  ……文字列100をすべて200に置き換える。
```

pretty printing コマンドはプログラムを清書し，端末あるいはファイルに出力する。このとき，字下りの字数，清書する手続き，定義・宣言部のみの清書，実行本体のみの清書，ファイルへの出力，1行に2つ以上の文を書き出さないなどのオプションを指定できる。

```
PP   sole pasout 3  ……字下りを3字とし，1行に1文を書き出し，ファイル pasout
                      にプログラム全体を清書して出力する。
PP   sub state only  ……手続き sub の実行本体のみを端末に出力する。
PP   sub dec only sole  ……手続き sub の手続き宣言を除く定義・宣言部のみを，1行に
                      2つ以上の定義・宣言をしないようにして清書する。
```

実行本体のみの編集は対象となる文のみの処理ですむので，実行途中であっても現行節が消失していなければ，そのまま実行を継続することができる。しかし，定義・宣言部および手続きの編集があった場合には，内部的には編集の影響が及ぶ範囲を翻訳し直さなければならず，復元指定いかんによっては途中から実行再開できる場合もあるが，再開できないときには始めからのやり直しになる。ただし再翻訳などの内部的処理をプログラマは一切関知する必要がなく，一連の編集コマンドの後の実行コマンドの直前に，再翻訳などの内部的処置が施こされる。

2.3 PENSES のもとでの作業例

図4に PENSES のもとでの虫取り作業の例を示す。この例では，対象となるプログラムから故意に変数 S の初期設定を削除してある。変数の動的表示により，S の異常に気づき，端末からの割り込みで中断させた後に，S の初期設定のための代入文の挿入，および **repeat** 文から

```
PENSES
  PENSES IS STARTED
?SO
  PASCAL PROGRAM SOURCE LIST
    10 PROGRAM PASCAL(OUTPUT);
    20  CONST C=50;
    30  TYPE PTR=@NODE;
    40       NODE=RECORD KEY:INTEGER; NEXT:PTR END;
    50  VAR S,N:INTEGER; P:PTR;
    60  PROCEDURE FACT(MAX:INTEGER);
    70   VAR Q:PTR;
    80  BEGIN N:=0; P:=NIL;
    90   REPEAT N:=N+1; S:=S*N;
   100    NEW(Q); Q@.KEY:=S; Q@.NEXT:=P; P:=Q
   110    UNTIL MAX<=S
   120  END;
   130 BEGIN FACT(C); WRITELN(N,S) END.
  NO ERRORS IN THIS PASCAL PROGRAM
   130.0   PROGRAM-NODE
?RSP FACT
?D * S,N
?R
     80.1: N <--          0
     90.1: N <--          1
     90.2: S <--          0

  INTERRUPTED
   100.0   NEW-NODE
?SIF 90 S:=1;
?SR 90 WHILE MAX>S DO BEGIN <90.1..100.3> END;
?RS
    80.0   PROCEDUR-NODE
?L N;
          0
?PP FACT SOLE STATEONLY 3

   PROCEDURE FACT(MAX:INTEGER);
   BEGIN  (* FACT *)
      N:=0;
      P:=NIL;
      S:=1;
      WHILE MAX>S DO
         BEGIN
            N:=N+1;
            S:=S*N;
            NEW(Q);
            Q@.KEY:=S;
            Q@.NEXT:=P;
            P:=Q
         END
   END;  (* FACT *)

?DC * S,N
?C
         5        120
   130.0   PROGRAM-NODE
?L P@.NEXT;
 #0 --> #1
 #1.KEY:         24
 #1.NEXT --> #2
 #2.KEY:          6
 #2.NEXT --> #3
 #3.KEY:          2
 #3.NEXT --> #4
 #4.KEY:          1
 #4.NEXT --> NIL
?CO C=10000;
?R
         8     40320
   130.0   PROGRAM-NODE
?E WRITELN('  DEMONSTRATION OF PENSES');
  DEMONSTRATION OF PENSES
?BYE
  PENSES IS TERMINATED
===
```

図4　PENSES のもとでの作業例（'?' で始まる行がコマンド入力である）

while 文への改造を行なっている。その後，手続きの先頭に復元し，それを N の値で確認し，手続きの清書を行ない，実行を再開して正常な結果を得ている。そのほか，動的なリスト構造の出力，定数の変更による再計算などを行なっているが，PENSES の強力な支援機能を見ることができよう。

3. PENSES におけるプログラム処理方式

PENSES は HITAC-M 200H の VOS 3 システムのもとで作成され，OS との情報の授受を除いて Pascal で記述されている。PENSES は内部手続きとしてコンパイラを内蔵しており，能率と制御性を両立させるため，機械語モジュールを直接取り扱っている。以下に主な処理手順について述べる。

3.1 木構造に基づいた実行制御

プログラムを木構造に展開したときの各節の役目は，基本的には式の評価と，代入節であれば左辺の変数の番地の評価である。内蔵のコンパイラはそのための機械語モジュールを各節に割り当てておく。PENSES は実行コマンドの指示に応じて，図5に示すように処理モード切換えル

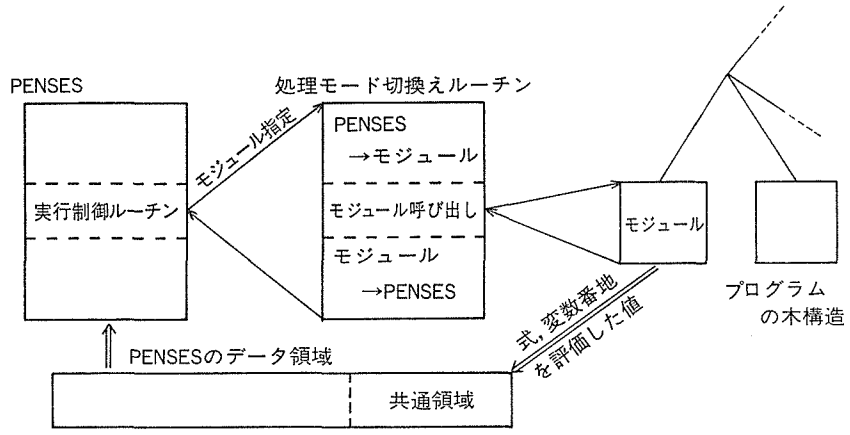


図5 処理モード切換えによる機械語モジュールの制御

ーチンに節の所有するモジュールの実行を指示する。このルーチンは PENSES 側とモジュール側のレジスタ類の待避と復旧を行ない，モジュールを駆動した後，再びモードの切換えを行なう。モジュール側で評価した値は共通領域を通して PENSES 側に渡される。PENSES 側では節の種別に応じて，代入節であれば，復元指定があるとまず旧データの待避をした後に更新処理を行ない，繰返しまたは分岐に対応した節であれば，式の値から次に実行すべき節を求めるなどの処理を行なう。

3.2 オペランド処理のための部分翻訳

PENSES はコマンドのオペランドに Pascal の構文規則に従う手続き，型，文，式および変数を受け付けるので，これらを翻訳しなければならない。そのため，図6に示すように，プログラムの各レベルの識別子表を手続き宣言および **with** 文の出現に応じて木構造にして所持しており，任意のレベルの節から根の方を見ると識別子表がスタック構造に見えるようになっている。プログラムの各節は，自身が位置づけられている識別子表を指しており，従ってプログラムの節

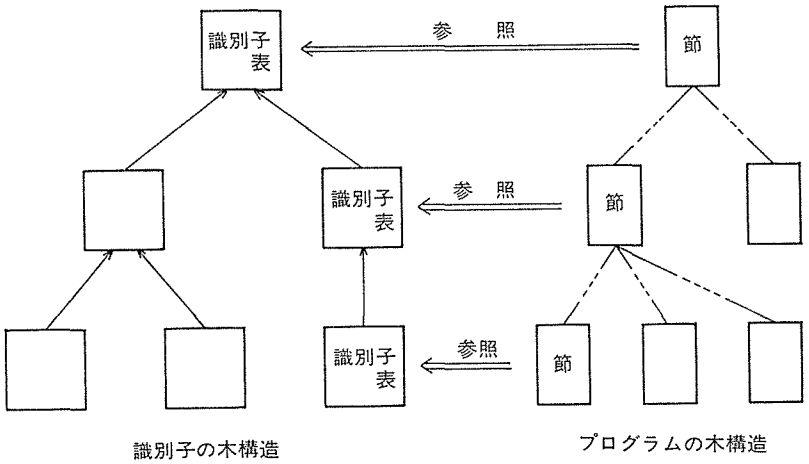


図6 節から木構造化された識別子表の参照

を指定すれば翻訳上の位置づけがわかる仕組みになっている。コンパイラは識別子表上の位置づけと翻訳すべき構文単位を指示されて部分翻訳を行なう。ただし，定義・宣言部の編集，および定義・宣言部の編集後の実行本体の編集のときは構文解析のみを行ない，実行コマンドの直前に再翻訳を行なう。

3.3 復元処理

手続きごとの復元指定に対しては，大域的変数および仮パラメータのみを手続き節に対応させて待避する。一方，節ごとの復元指定に対しては，図7に示すように，まず並置関係，次いで入れ子関係をさかのぼっていくので，自身の節を含め，親の節をたどって旧データの待避を行なう。ただし，同じ変数に対するデータがすでに待避されていたら，そこで待避処理を打ち切る。なお，手続きの再帰的呼び出しに備え，呼び出しごとに待避データをスタックしておく。待避データはモジュール側から受け取る変数の割り当て番地，データ領域の大きさおよび旧データから

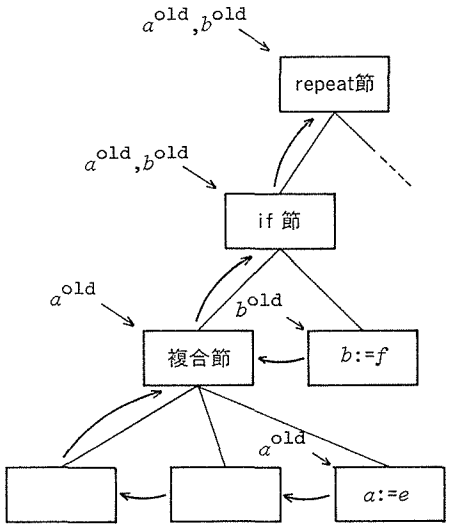


図7 データ待避と復元経路

なり，前二者で同一変数に対するものであることを識別する。変数が大域的変数または変数型の仮パラメータのときには，呼び出している手続きに復元指定のものがあれば呼び出しの結合をさかのぼり，必要なデータの待避を行なっていく。復元指示があったときには，最も近くの復元可能な節を探し，その節が所持している待避データからデータ領域の復元を行なう。

3.4 変数の動的出力とダンプ出力

動的出力のために，代入文に対応する節にはあらかじめ出力文を割り当ててある。一方，識別子表の変数名は代入文の左辺に現われた節を把握しており，指示に応じて出力文を実行するか否かを示すフラグを設定する。list コマンドによる変数のダンプ出力は次のような手順で行なう。まず，オペランドの変数を翻訳して，変数の割り当て番地を計算するモジュールを作成する。この時，その変数のもつ型も求まる。次に，作成したモジュールを駆動して変数の番地を求める。求まった番地からは，先の翻訳時に得られた型に対応したデータが入っており，型の種別に応じた出力ルーチンを働かせる。ただし，ポインタ型のデータは動的変数を指している番地であるので，動的変数の型に対し再び出力ルーチンを再帰的に駆動する。変数がレコード型の場合，ループ状のデータ構造を形成することがあり，上述の手順では無限ループに入ってしまう。これを避けるためには各動的変数に出力ずみのマークを付ければよいのであるが，そのためには初期設定を必要とし，そのためにやはりループしてしまう。そこで，list コマンドが使われた回数をカウントし，その値を出力ずみ変数に設定していき，設定値とカウンタとの一致をとって既出の判定を行なっている。

4. おわりに

PENSES の前身である，実行制御のみを行なうシステム INSIDER から 2 年以上の使用経験があり，その強力な支援機能を十分に発揮することが実証されている。現在のシステムはタイプライタ型の端末を使用しているが，今後 CRT スクリーンを対象とし，よりマン・マシーン結合をよくし，なかんずく診断データを図形的に表示するようにシステムの改良を行なっている。

参考文献

1) 鳥居宏次，二木厚吉，真野芳久：情報処理，20, 1, p. 22～43 (昭54)
2) Conway, R. W. and Wilcox, T. R.: Commun. ACM, 16, 3, p. 169～179 (1973)
3) 原田賢一，Zelkowitz, M. V.: 情報処理，16, 2, p. 85～92 (昭50)
4) 藤村直美，牛島和夫：情報処理，17, 6, p. 547～550 (昭51)
5) 春原猛ほか：情報処理，20, 5, p. 405～411 (昭54)
6) 宮本衛市，浅見可津志：情報処理，20, 6, p. 474～480 (昭54)
7) Hart, J. J.: SIGPLAN Notices, 14, 12, p. 110～121 (1979)
8) Cichelli, R. J.: SIGPLAN Notices, 15, 1, p. 34～44 (1980)
9) 宮本衛市，堀川博史，高橋幸伸：情報処理，22, 5, p. 424～433 (昭56)
10) Shapiro, E. *et al.*: SIGPLAN Notices, 16, 8, p. 50～57 (1981)
11) Atkinson, L. V. and North, S. D.: Software-Practice and Experience, 11, 8, p. 819～829 (1981)
12) Brown, P. J.: *ibid.*, p. 831～843
13) Oldehoeft, R. R.: *ibid.*, p. 867～873
14) Teitelbaum, T. and Reps, T.: Commun. ACM, 24, 9, p. 563～573 (1981)