構文指導型プログラム開発システムについて

海　尻　賢　二
（信州大学工学部）

１．はじめに

最近プログラム開発におけるツールの重要性が叫ばれている。コンパイラ，エディタといった従来からあるツールに加えて，インタフェイスチェッカ，静的解析ツール，動的解析ツール，シンボリックデバッカ等多くのツールが作られ，そして実際に使われている。これらのツールを使うことによりプログラム開発の効率は著しく向上する。そのような観点から従来のプログラミング言語とそのコンパイラというシステム構成ではなく，プログラミング言語とそのツール一式というシステム構成でプログラミングシステムというものを考えるようになってきている。そのようなシステムをプログラミング環境（ＰＥ）と呼ぶ。Adaのプログラミング環境ＡＰＳＥが有名である。

ＰＥの構成としては上で述べたツール群の集合体という形態では不十分であり，ツール群の有機的な複合体という形態が望ましい。このようなＰＥは図１に示すように共通のデータベース（プログラムの内部表現）に対して作用する統合されたシステムという形態をとる。そのようなＰＥとしては，Cornell大のCornell Program Synthesizer〔２〕，ＣＭＵのＩＰＥ〔３〕，ＩＢＭのLispedit〔４〕，ＩＲＩＡのMentor等がある。

統一された複合体としてのＰＥの持つべき機能としては次のようなものが考えられる。

①入力と同時の構文単位毎の構文解析と，それの解析木への埋め込み
②構文指導型のプログラミング（syntax directed programming）
③スクリーンエディト
④解析木に対する木エディット
⑤プログラムのスクリーンへの圧縮表示（screen pretty printing）
⑥スクリーンを利用した実行のモニタ
⑦不完全なプログラムに対しても実行可能なインタプリタ（incremental interpreter）
⑧静的及び動的解析
⑨プログラム要素の維持，管理
⑩各種文書の自動生成

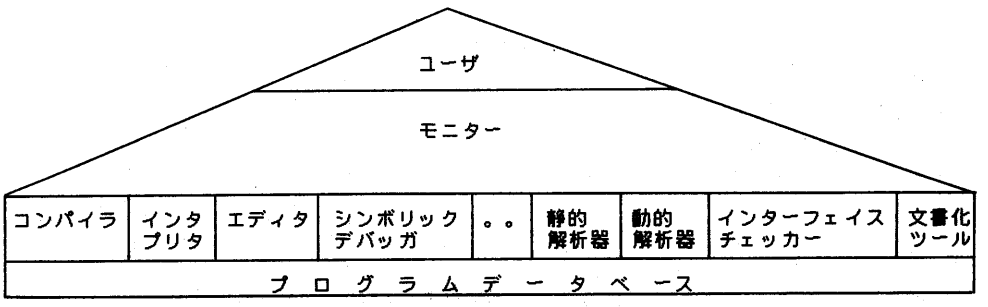

図１。　統合されたプログラミング環境
Fig. 1　Unified Programming Environment

仕様が与えられた時の入力，編集，試行というミクロな開発環境に焦点を絞って考えるならば，仕様に対する検証，要求仕様の記述などは考えなくてもよい。上記の各機能は個々には既に実現され，実用に供されているものもある。本論文でもこのミクロな開発環境に焦点を絞り，上記の機能の中で特に①，②，④～⑦の機能を統一したデータベース（解析木）のもとで実現した構文指導型プログラム開発システム（ＳＤＰＤＳ）について述べる。

以下２章ではＳＤＰＤＳの概要を，３章ではその実現を述べ，４章で評価，検討を行なう。

２．ＳＤＰＤＳの概要

ＳＤＰＤＳは構文指導型の対話型ＰＥであり，特定の言語によるプログラム開発における次の３つのステップをサポートする。

①詳細仕様からのプログラム作成
②試行
③編集，虫取り

これらの途中結果はスクリーンの画面内におさまるように圧縮して表示される。システムは入力及び編集のための構文指導型エディタ，スクリーンへの圧縮表示のためのスクリーンプリティプリンタ，そして試行のためのインクリメンタルインタプリタの３つより成る（図２）。この内インタプリタを除く２つは表駆動型で実現されており，言語独立である。入力はテンプレートに基づく構文指導型入力，編集は解析木を対象とした木エディット，試行は解析木に対するインクリメンタルインタプリタにより各々行なう。実現したプロトタイプはWirthのＰＬ／に入出力文を追加した言語を対象としている。この言語の文法定義を図３に示す。この文法定義の右端の記述はプリティプリンタのための情報である。

ＳＤＰＤＳで対象とする言語の構文はSynthesizerの流儀にならい，テンプレート型構文，繰り返し型構文，並列型構文，フレーズ型構文の４つに分類する。テンプレート型構文はその中に予約語を含む構文である．図３ではprogram，block，const-dec，var-dec，pro-dec，calls，compound，ifs，whilesがそうである．繰り返し型構文とはある終端語をはさんで繰り返される非終端語を表わす構文である。図３ではstate-list，cexp-list，ident-list，exec-listがそうである．図３において

A::=〔ＢＣ〕

は

A::=B｜BCA

の意味である。並列型構文とは非終端語のalternativeからなる構文である。図３ではstatement，exec-stateがそうである．フレーズ型構文とはそれ以外のものである。図３では特に下半分に区別して記述してある。記述上は

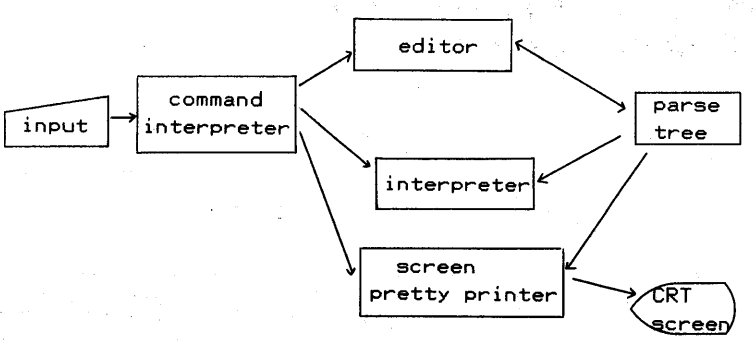

図２。　構文指導型プログラム開発システム
Ｆｉｇ。２　Ｓｙｎｔａｘ　Ｄｉｒｅｃｔｅｄ　Ｐｒｏｇｒａｍ　Ｄｅｖｅｌｏｐｍｅｎｔ　Ｓｙｓｔｅｍ

何の制限もない。SDPDSの対象とする言語の構文名はどれか1つのクラスに属す必要がある。

```
    ******** template type syntax ********
program       ::=block.                           1'.'
block         ::=BEGIN state-list END             'begin' cr
                                                  2X 1 cr
                                                  'end'
state-list    ::=[statement ; ]                   1 ';' cr
statement     ::=const-dec
              ::=var-dec
              ::=pro-dec
              ::=exec-state
const-dec     ::=CONST cexp-list                  'const ' 1
cexp-list     ::=[cexpression , ]                 1 ','
var-dec       ::=VAR ident-list                   'var ' 1
ident-list    ::=[ident , ]                       1 ','
pro-dec       ::=PROCEDURE ident ; block          'procedure ' 1 ';' cr
                                                  2X 2
exec-state    ::=pstatement
              ::=calls
              ::=compounds
              ::=ifs
              ::=whiles
calls         ::=CALL ident                       'call ' 1
compounds     ::=BEGIN exec-list END              'begin' cr
                                                  2X 1 cr
                                                  'end'
exec-list     ::=[exec-state ; ]                  1 ';' cr
ifs           ::=IF condition THEN exec-state     'if ' 1 'then' cr
                                                  2X 2
whiles        ::=WHILE condition DO exec-state    'while ' 1 ' do' cr
                                                  2
    ******** phrase type syntax ********
pstatement    ::=assigns
              ::=inputs
              ::=outputs
assigns       ::=ident:=expression
inputs        ::=INPUT [ident ,]
outputs       ::=OUTPUT [expression ,]
condition     ::=expression <= expression
              ::=expression <  expression
              ::=expression == expression
              ::=expression ^= expression
              ::=expression >= expression
              ::=expression >  expression
expression    ::=- [term addop]
              ::=[term addop]
addop         ::=+   -
term          ::=[factor mulop]
mulop         ::=*  /
factor        ::=ident
              ::=number
              ::=( expression )
cexpression   ::=ident=number
```

図3。　PLOの構文定義
Fig。3 Syntax Definition of PLO

テンプレート型構文の入力はテンプレート名の入力により行なう。例えばIf文の入力にあたっては@Ifsと入力する。すると

IF condition THEN
　　exec-state

というIf文のワク組み（テンプレート）が解析木の中へ組み込まれる。フレーズ型構文はそれを構成する文字列をすべて入力（つまり普通の入力）することにより行なう。繰り返し型構文の入力は後に述べるreturn命令により行なう。このテンプレート形式による入力は次のような特徴を持つ。

①キーストロークが少ない。テンプレート名を指定すればよいのであるから長い予約語（たとえばsubroutine）のようなものの入力もより簡単に行なえる。特定のキーをテンプレート名に対応付けておけば1キーで代用させることも可能である。

②Syntactic sugar的な構文要素の入力の必要がなく，文法の修得が容易である。If文においてThenはsyntactic sugarである。プログラムの見やすさからは重要なものではあるが，入力又は言語の修得という面からは面倒なものである。このような要素はテンプレート方式では憶える必要も，入力の必要もない。又Ifの次がconditionでThenの次がexec-stateであることはスクリーンに表示されるため，細い構文は憶える必要がない。

このようにテンプレート方式は多くの利点を持つが，欠点は少し言語に慣れると入力がかえって面倒になるという点である。そこでIPEのようにすべての入力をテンプレート方式で行なう方式は採らず，条件式とか算術式等はそのまま入力するSynthesizerの方式を採用した。

図4にPLØによるプログラム例，図5にその内部表現である解析木の形態を示す。

テンプレート型構文及び繰り返し型構文の構成要素である非終端語を，各々その成分と呼ぶ。例えば図3においてIf文の成分はconditionとexec-state，exec-list

```
BEGIN
 IF A<=B+10 THEN
  CALL identifier;
 WHILE A==B DO
  BEGIN
   A:=B+10
  END;
 A:=B*2
END.
```

図4。　PLOのプログラム例
Fig。4 An Example of a PLO Program

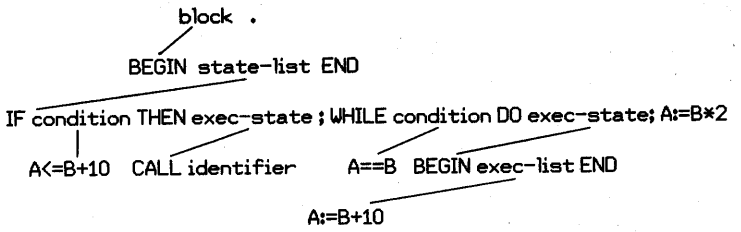

図5。　解析木の例
Fig。5 An Example of a Parse Tree

の成分はexec-stateである。解析木のノードはテンプレート型構文又はフレーズ型構文である。テンプレート型構文はいくつかの成分を持つが，この成分の具体値をそのテンプレート型構文の子供と呼ぶ。即ちテンプレート型構文IFSの成分conditionの子供という表現になる。例えば図5においてA==Bはテンプレート型構文whilesの成分conditionの子供である。又繰り返し型構文の成分間の関係を兄弟と呼ぶ。例えば図5ではwhilesの左の兄弟はifs,右の兄弟は代入文である。完成したプログラムにおいては解析木のリーフノードは必ずフレーズ型構文であるが，未完のプログラムではテンプレート型構文もリーフノードになり得る。

SDPDSにおけるプログラム作成は解析木のplaceholderと呼ぶ特別な部分への，構文の挿入，削除，置換により行なう。Placeholderとは解析木において変更可能な部分であり，テンプレート型構文及びその成分，繰り返し型構文及びその成分，そしてフレーズ型構文が成り得る。例えばIF文全体は1つのplaceholderとなり得るが，予約語IFやTHENはplaceholderとはなり得ない。condition，exec-stateは1つのplaceholderである。このようにSDPDSの入力，編集方法では予約語の入力誤りはあり得ない。

SDPDSのコマンドを図6に示す。コマンドはすべて@で始まる。@で始まらない文字列はフレーズ入力とみなされる。これらのコマンドの内テンプレートコマンドのみが言語に依存している。実行時のディスプレイ配置を図7に示す。プログラムはリスト領域に圧縮表示される。下の2行の領域はユーザからの入力及びシステムからのメッセージの領域である。プログラムリストの各行は左端に行番号を持つ。

次にSDPDSの各種の動作を詳しく述べる。

a. プログラム入力

プログラムの入力はplaceholderへのテンプレート型構文又はフレーズ型構文の入力，及びmove命令によるplaceholderの移動により行なう。こ

```
1. ******* template command *******
   PROG    program
   BLOK    block
   COSD    const-dec
   VARD    var-dec
   PROD    pro-dec
   CALL    calls
   COMP    compounds
   IFS     ifs
   WHIL    whiles
2. ******* move commmand *******
   UP              PF1
   DOWN            PF2
   LONGUP          PF3
   LONGDOWN        PF4
   DIAGONAL        PF5
   TOP
   MOVE n          (1-20)
   (return)
3. ******* program edit command *******
   CLIP            clip the current subtree into worktree
   CLIP fname      clip the current subtree into file fname
   COPY            copy the current subtree into worktree
   COPY fname      copy the current subtree into file fname
   INSERT          insert from worktree
   INSERT fname    insert from file fname
4. ******* program transfer command *******
   SAVE fname      save program into file fname
   LOAD fname      load program from file fname
   LIST fname      pretty print program into file fname
   LIST            pretty print program into CRT-screen
   NEW             program area clear
5. ******* interpret command  ****************
   RUN             incremental interpretation
```

図6。　SDPDSのコマンド一覧
Fig。6 SDPDS Command List

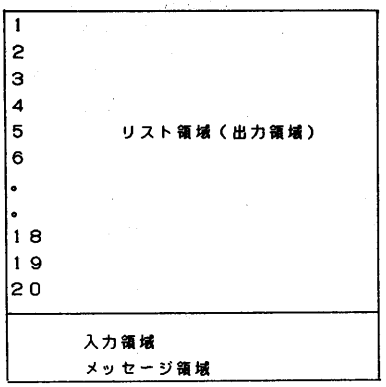

図7.　SDPDSのスクリーンの配置
Fig. 7 SDPDS Screen Layout

の入力と編集を行なうモードを編集モードと呼ぶ。
システムは初期状態ではテンプレート型構文 program を受け入れる状態にある。そこで@prog と入力する。すると

1　＊ block.

とスクリーンには表示される。スクリーン表示する時，placeholder は色をかえて表示する。紙面では下線で色のかわっている部分を示す。テンプレートがplaceholderの時はその先頭の予約語の色をかえる。先頭に予約語を持たない時（上例の場合）は＊という特別の記号を先頭に追加し，placeholder の位置を示す。Placeholder がこの位置では修正しかできないので move 命令 @down により placeholder を1つ下げる。すると

1　block.

のように表示される。この時点で入力できるのはブロックだけである。そこで@blok と入力する。すると右の図8(a)のように表示される。このようにして図8(b)のようになったとする。condition はフレーズ型構文であるのでここでは直接に condition の具体値を

ab >= a

のように入力する。すると図8(c)のようにかわる。

```
1  BEGIN
2    state-list
3  END.
     (a)

1  BEGIN
2    IF condition THEN
3      exec-state
4  END.
     (b)

1  BEGIN
2    IF ab>=a THEN
3      exec-state
4  END.
     (c)
```

図8.　例
Fig. 8 Example

テンプレート名が正しくなかった時（構文的に対応しなかった時），又フレーズ型構文として入力した記号列が構文的に正しくなかった時はスクリーンのメッセージ領域に誤りメッセージが表示される。この時プログラムは変化しない。即ち文法的に正しい入力でない限り入力は受け付けられない。このようにSDPDSによるプログラム作成においては，作成の途中においてもそのプログラムは文法的に正しい（但し非終端語を含む）。他の move 命令の動作を図9に示す。図9においてワクで囲んだ部分が placeholder であり，＿＿のように表示されている部分が各move命令により新たにplaceholderとなる部分である。

Return命令（入力要求？に対してreturnキーで答える）は繰り返し型構文の成分の挿入に使用する。図9(a)におけるようにplaceholderが繰り返し型構文の成分（state-listの成分 exec-state の alternative の1つが whiles）である時，return 命令を入力するとwhiles と代入文の間に statement が挿入されて，図9(b)のようになる。この statementに対応するノードは解析木上で挿入のためだけに用意されたものであり，dummy node と呼ぶ。Dummy node に対して挿入が行なわれずにplaceholder が移動させられると dummy node は消去される。つまり図9(b)で@down と入力すると statement は消去され，図9(a)の状態へ戻る。但し placeholder は代入文 B:=A+1 へ移る。Dummy node の挿入ができないノードでの return命令は@down と同じ働きをする。即ち図9(b)における return 命令は@down と同じ働きとする。

Placeholder はこのようにmove命令により移動させることができる．しかしこのような命令だけではplaceholderの大きな移動には不十分である．カーソルをスクリーン上で移動させてplaceholderを指定するという方法もあるが，SDPDS ではスクリーンのリスト領域の行番号を利用して

@ move n

のように移動すべき行を指定するという方法を採用した．例えば図9(a)で

@ move 8

とすると代入文 A:=B へplaceholderが移動する．

```
1  BEGIN
2    PROCEDURE identifier
3      BEGIN
4        IF condition THEN
5          A:=A+10;
6        WHILE A==0 DO
7          BEGIN
8            A:=B;
9            .....
10         END;
10       statement;
11       B:=A+1
12     END;
13   .....;
14 END.
```

(b)

```
1  @UP
2  @DOWN
3  @LONGUP
4  @LONGDOWN
5  @DIAGOBNAL
6  @TOP
```

(a)

図9。 move命令の動作
Fig。9 Action of Move Commands

b. プログラムの修正

プログラムの修正はmove命令によるplaceholderの移動と3種類の編集命令で行なう．本システムはプロトタイプであるので編集命令としては解析木に対する3種類の木エディット命令（複写－COPY，削除－CLIP，及び挿入－INSERT ）のみを持つ．挿入は削除又は複写した部分木の挿入である．挿入にあたっては構文的に適合しないと許可しない．

C. インクリメンタルな解釈，実行

SDPDS のような開発システムではプロトタイプに基づく検査ツールが不可欠である．SDPDSはそのために解析木中のテンプレート型構文のすべての成分の具体値を必ずしも必要としないようなインタプリタを持つ．このインタプリタは命令@RUNにより起動される．この命令によりシステムは編集モードから実行モードへ入る．実行モードにおいてもスクリーンプリティプリント機能は生きており，次の各時点でプログラムをディスプレイに表示する．

①入力文による入力要求時
②具体値を持たない成分の実行時

②の場合は一担実行が中断され，以下のコマンドにより，その実行の制御，状態の把握が可能である．

a.	? name	その時点で有効な変数 name の値を出力する．
b.	name, number	その時点で有効な変数 name の値としてnumberを与える．
C.	number	値 number を条件式の値として代入する．対象が条件式でない時は無効である
d.	空	実行を再開する．
e.	@	実行モードを中断して，編集モードへ復帰する

現在実行モードは一種類しかなく，ワンステップ実行等は用意していない．実行モードでの出力はリスト領域へ表示される．実行モードでのプログラムの入力，編集はできない．

d. その他

@save fname は解析木を永久ファイルfnameへ格納する．@load fname は永久ファイルfname より解析木をロードする．SDPDS では通常の場合プログラムはスクリーンのサイズ（20行）内でしかみられない．そこでこの条件を無視して全体を見たい時には@list（又は@list fname）とする．するとスクリーンに通常の形態で出力される（fnameを付けると永久ファイルfnameへも同時に出力される）．@newは解析木の領域をクリアして，初期状態へ戻す．

3. システムの実現

SDPDS のハードウェア構成を図10に示す．ACOS 600S TSS 上での実現であるが，端末に関しては特別な画面制御を必要とするため，マイクロコンピュータ Hitachi レベル3を使用し，その端末化のプログラムは8ビットCPU6809用簡易コンパイラ言語 COMSOL で作成した．記述言語は構造化FORTRAN, RATFOR-R である．

以下ではスクリーンプリティプリンタ，エディタ，インクリメンタルインタプリタについて，その実現法等の詳細を述べる．

3. 1 スクリーンプリティプリンタ

SDPDS におけるスクリーンプリティプリンタはどんなに長いプログラムであっても何らかの形でスクリーン内におさまるように表示することを目的として設計したもので，次の特徴を持つ．

①どんなに長いプログラムであってもplaceholderの近傍を圧縮して，スクリーン内におさまるように表示する．

②表駆動型であり，その表は図3の右端に書いてあるような構文毎の清書パターンから自動的に作り得る．

スクリーン用プリティプリンタとしてはsynthesizer におけるようなユーザの制御によるものと，〔1〕におけるような現在の位置と回りの構造から自動的に行なうものとがある．SDPDS は後者の方式を採用している．

図3の構文定義の右端の清書パターンについて簡単に述べる．スクリーン用プリティプリンタに対して必要となる情報は各構文（実際にはテンプレート型及びフレーズ型構文）の清書スタイルとその必要とする行数である．清書パターンは

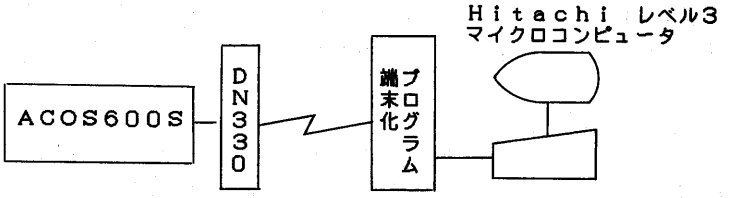

図10. SDPDSのハードウェア構成
Fig. 10 Hardware of SDPDS

この2つの情報を含んでいる。Compoundsに対する記述

```
compounds ::= BEGIN  exec-list  END      'begin' cr
                                         2X 1 cr
                                         'end'
```

を例として詳しく述べる。この記述はcompoundsを次のように清書することを指示している。

```
begin
    exec-list
end
```

清書パターン中の'(single qucte)で囲まれた記号列は直接の出力文字を，nxはn個の空白を，crは改行を，そしてiは生成規則の右辺のi番目の非終端語の清書をそれぞれ指示する。又compoundsの清書に要する行数は清書パターン中のcrの数+1で決まる。

SDPDSのプリントアルゴリズムは3パスより成る。パス1，2で内部表現である解析木の各ノードにflagを立て，パス3でそのflagに基づきスクリーンにプリントする。flagは次の4種類の値を持つ。

1……そのノードの構文を出力する。
2，3…省略記号又は構文名で出力する。
4……無視する。

圧縮型の表示においては親，兄弟，子供の3者の表示における優先順位が問題となる。SDPDSでは親，子，兄弟の順に優先順位を与えてある。

パス1.

placeholderのflagを1に，その左の兄弟を2に，右の兄弟を3に設定する。次に親へ移り同じことを繰り返す。これはplaceholderをリーフとして，その上部構造に重点をおいて表示することを意味する。

パス2

このパスはパス1の実行後，まだスクリーンにプリントできる行が残っている時に実行される。まずplaceholderの子供にflagを立ててゆき，次にパス1で2，3としたノードのflagを1にしてゆく。又flagの値が4のノードも2，3へ，そして1へしてゆく。パス2はパス1によって決った解析木の骨組みに対する肉付けに相当する。

パス3

パス1，2で立てられたflagに基づきスクリーンに清書する。その際スクリーン上の行番号とプリントした解析木のノードとの対応を憶えるノードマップを作成する。

図11に20行から成るプログラムを14行におさめた時の表示例をplaceholderの位置毎に示す。図11(a)はもとのプログラム(@listによる出力)であり，(b)，(c)，(d)が各々下線部がplaceholderの時の表示である。

3.2 エディタ

SDPDSのエディタはプログラムの入力及び編集を受け持つ。フレーズ型構文を除きすべて表駆動形式であり，図3の構文定義から自動的に作られる。(但し現在はまだGeneratorは実現していない)・フレーズ型構文は再帰下降型パーサにより解析され，語い解析後のトークン列の形式で解析木へ組み込まれる。

```
begin
  var X,Y,Z;
  procedure GCD;
    begin
      var F,G;
      F:=X;
      G:=Y;
      while F^=G do
        begin
          if F<G then
            G:=G-F;
          if G<F then
            F:=F-G
        end;
      Z:=F
    end;
  input X,Y;
  call GCD;
  output X,Y,Z
end.
```
(a)

```
1  ¥begin
2      var X,Y,Z;
3      procedure GCD;
4        begin
5          var F,G;
6          F:=G;
7          G:=Y;
8          while F^=G do
9            begin
10             exec-list
11           end
12       end;
13     ........
14   end.
```
(b)

```
1   begin
2     var X,Y,Z;
3     procedure GCD;
4       begin
5         ........;
6         while F^=G do
7           begin
8             if F<G then
9               G:=G-F;
10          if-statement
11        end
12      end;
13    ........
14  end.
```
(c)

```
1   begin
2     var X,Y,Z;
3     procedure GCD;
4       begin
5         var F,G;;
6         F:=X;
7         G:=Y;
8         while F^=G do
9           exec-state
10      end;
11    input X,Y;
12    call GCD;
13    output X,Y,Z
14  end.
```
(d)

図11。　表示例
Fig。11　Examples of Pretty Print

3．3　インクリメンタルインタプリタ

インタプリタはフレーズ型構文の解析と同様再帰下降型を採用した。但し式の評価はポーランド記法を利用して行なった．具体値を持たない成分の実行に到った時は optional input mode に入り，各種の命令を受け付ける。Dummy node については無視して実行を続ける。

4．SDPDS の評価

筆者は種々の形態のプログラミング環境を実際に実現して，その評価を行ない，個人的な環境における理想的なPEを追求している．そのような観点からSDPDSはテンプレート型入力による構文指導型エディタ及びスクリーン用プリティプリンタについて考察するという目的から作成した。そしてPEにおいてスクリーンの特性をできるだけ生かすという方針のもとで設計した．

次に特に問題となった点について考察を行なう．その1はスクリーン用プリティプリンタである。SDPDSで考案した方法は〔ユ〕でのboxという概念に基づく方法とは異なり，解析木をある前もって決めた優先順位に従って走査し，flagを立てていき，最後にそのflagに基づきプリントを行なうというものである．ある時点でスクリーンの行数を越えるとそこで走査を打ち切るので，スクリーンの行数を越えることはない．又優先順位の変更により圧縮の基本方針が簡単に変えられるという特徴を持つ．実現したシステムではスクリーンの行数を20としているが，これは使ってみるとやや少ない。40～50行表示できる安価なCRTが望まれる．このようなプリント方式はwindow方式のプリンタと併用することによりその効果が上がる．それを使いわけることをユーザが指示できることが望ましい．SDPDSでのスクリーンプリンタでは優先順位の変更によりwindow方式も容易に実現できる．

その2はスクリーン上でのフルスクリーンエディットである。マイコンのBASICで行なえるような機能が望まれるが，TSSで実現しており，ホスト側のTSS制御プログラムの作成は困難であったため今回はあきらめ，本エディットのみにとどめた．現在SDPDSのマイコン版を計画中であり，そこではフルスクリーンエディットを実現する予定である。

その3はテンプレート方式の入力である．テンプレート方式の入力法は慣れると入力が面倒になる．しかしこれは言語に習熟してしまっていることによる．初心者が端末を前にして言語を修得してゆくと考えると十分に有用な方式である．た

だこのような点からの評価はまだ行なっていない．

オ々はインクリメンタルインタプリタである．初心者用のＰＥでは実行して結果を見るということが重要な要素となる．そのような点から不完全なプログラムでも実行可能なインクリメンタルインタプリタが望ましい．ここでもやはりBASIC的direct statementや，実行と編集の同時進行が望まれるが，本システムはそこまではできず，部分的なdirect statement及びスクリーン上での実行のモニタにとどまった．オーのプリティプリンタでもそうであるがここではよりスクリーンの狭さが問題となった．プログラムの圧縮リストと実行結果が同時にスクリーンに表示されればよいのだが，スクリーンの狭さから実現できなかった．その点から(5)の複数画面によるユーザインタフェイスが有望である．

5. むすび

本論文で述べたSDPDSはプロトタイプであり，実際に使ってみることによる改良が必要である．使ってみると入力即表示という形式は非常にわかりやすいものである．又SDPDSにおけるスクリーンプリティプリントは圧縮表示であるので，単にある行数におさめるということだけでなく，プログラム構造の理解にも役立つものである．他のシステムへの適用も有用と考える．

SDPDSはTSS上で作成したために不十分な点が多い．現在マイコンPC 8801上でも計画中であるが，そこではスクリーンの機能を十分に生かしたシステムにする予定である．次のような機能を予定している．

① PC 8801はテキスト画面とグラフィック画面を別々のディスプレイに表示できる．この機能を使って複数画面を実現する．

② PC 8801の漢字機能を使って応答を日本語で行なう．

このような機能の実現により，安価な個人用プログラム開発ワークステーションを目指す．

参考文献

(1) M. Mikelson : Prettyprinting in an Interactive Programming Environment, Sigplan notices Vol. 16 No. 6 (1981. 6)

(2) T. Teitelbaum : The Why and Wherefore of the Cornell Program Synthesizer, Sigplan notices Vol. 16 No. 6 (1981. 6)

(3) Peter H. Feiler : An Incremental Programming Environment, 5th Int. Conf. on S.E. 1981

(4) C. N. Alberge : A Program Development Tool, 8th Symp. on Principles of Programming Languages 1981

(5) 真野他 : 複数画面プログラミング環境における画面エディタＭＤＥＤ ソフトウェア工学 22-6 (1982 2)