最適化出力を生成する構造的 FORTRAN プリプロセッサ†

都司達夫†† 池羽田篤†† 渡辺勝正††

構造的 FORTRAN プログラムは従来のプリプロセッサにより通常1パスで標準的な FORTRAN プログラムに変換される．この場合の問題点は構造化文の変換のためにプリプロセッサが多くの冗長な continue 文および goto 文を生成することである．このことは，①変換後のプログラムが非常に読みにくい，②プログラム格納に要するファイルサイズの肥大，③ FORTRAN コンパイラおよびプログラムそのものの実行時間の増大，といった生産性の低下やプログラムの質の低下を招きうる．筆者らの知る範囲では従来のプリプロセッサはほとんど，このような冗長度を伴う1パスの方法で作られていると考えられる．本論文では，基本的に1パスでこれらの冗長性を抑制した最適化出力を生成するためのプリプロセッサの方式を新しく提案した．さらに，この方式に従って，構造的 FORTRAN 言語の一つである Westran を対象としてプリプロセッサを作成し，従来の方法によるプリプロセッサと比較して，種々の評価を行った．この結果，ユーザから見た処理系の実行時間である“プリプロセス時間＋FORTRAN コンパイル時間”は，従来の1パス処理に比して，むしろ速くなることを確認した．すなわち，冗長な文のファイル入出力を抑制することは，両方の時間の減少に有効であり，これは最適化のための時間的なオーバヘッドを十分補償しているためである．

1. まえがき

構造的 FORTRAN 言語は，FORTRAN がもっている数々の現実的な長所をそのまま引き継ぐとともに，構造的プログラミングを支援するための制御構造を FORTRAN の仕様に縛られることなく，統一的に与えることができる[1)~4)]．今後，新しい FORTRAN 言語仕様が登場してきても従来の仕様と互換性を保とうとする限り，統一的で大幅な仕様の改革は無理であり，この意味で構造的 FORTRAN の有用性は変わらないと考えられる．

構造的 FORTRAN プログラムは従来のプリプロセッサにより，通常1パスで標準的な FORTRAN プログラムに変換される．この場合の問題点は，構造化文の変換のために付録の(C)に見られるようにプリプロセッサが多くの冗長な continue 文および goto 文を生成することであり，このことは，①変換後のプログラムが非常に読みにくい，②プログラム格納に必要なファイルサイズの肥大，③ FORTRAN コンパイラおよびプログラムそのものの実行時間増大，といった生産性の低下やプログラムの質の低下を招きうる．筆者らの知る範囲では従来のプリプロセッサはほとんど，このような冗長度を伴う1パスの方法で作られていると考えられる．筆者らは，プリプロセッサによってすでに FORTRAN に変換されたプログラムを対象として，このような冗長性を除去するためのツールを提案したが[5)]，ここではプリプロセッサそのものが冗長性のない最適化出力を生成するための方式を新しく提案する．さらに，それに基づいてプリプロセッサを作成し，種々の評価を行ってその有効性を確認した．本プリプロセッサは構造的 FORTRAN 言語の一つであり，比較的洗練され，種類も豊富な構造化文をもつ Westran[2)] を対象言語として*，それ自身 Westran で書かれている．

2. 準備的な考察

従来のプリプロセッサを使用して1パスで FORTRAN コードを生成する場合には，通常構造化文の翻訳は**図1**(B)に見られるように，プリプロセッサが生成した文番号をもつ continue 文を発生する必要がある．なぜならプリプロセッサが生成する goto 文の参照先がソースプログラム中で文番号をもつか否かは goto 文生成時点ではわからないからである．構造的にプログラミングしたときには，滅多に使用しない文番号のために冗長な continue 文を発生することは好ましくない，避けるべきだと考えられる．たとえば，図1(B)の変換出力から continue 文を除去すれば図1(C)が得られる．このとき，図1(B)の③における 10 のように continue 文の次行にソースプログラムの文番号がつく場合には continue 文を参照している箇所①，②を 10 に書き換えておく．さらに，このように

† A Structured FORTRAN Preprocessor which Generates Optimized Output by TATSUO TSUJI, ATSUSHI IKEBATA and KATSUMASA WATANABE (Department of Information Science, Faculty of Engineering, Fukui University).

†† 福井大学工学部情報工学科

* Westran 本来の仕様には Pascal ライクなデータ構造の取扱い機能が含まれるが，本論文で説明するプリプロセッサではそれらをサポートしていない．

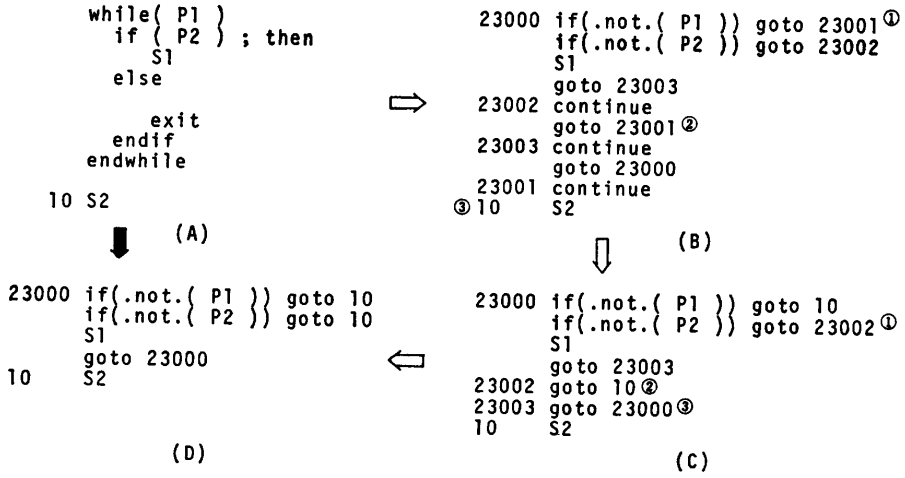


図 1 冗長性のないプログラムを得る過程
Fig. 1 The stages to get non-redundant program.

して冗長な continue 文を除去したならば，図1(C)の②,③の goto 文は直前が goto 文であるために除去可能であり，図1(D)が得られる．このとき，たとえば図1(C)の①における23002は10に書き換えておく．

ここでは，冗長な continue 文と goto 文の発生を抑制するのに，図1(B),(C)の段階を経ることなしに直接，図1(D)の最適化出力を得るための方法を考える．このような目的のためには，一般に2パス方式のプリプロセッサが必要になるが，ソースファイルを2度読みすることは，処理時間の点で明らかに不利である．そこでここでは構造化文用の予約語に対する変換結果として前方参照が必要な goto 文および goto を含む論理 if 文を出力する必要がある (if, while 等) とき，とりあえず暫定的な文番号を発生して参照先としておき，それを内部表に登録するとともに構文解析用のスタックにもそれを格納しておく．同時に内部表にはそれらの文のレコードの出力ファイルにおける位置も合わせて記録しておく．以後，この暫定参照文番号を goto で参照する必要のある予約語が現れたときには，構文解析用のスタックから先に格納した文番号を引き当て，やはりそれと goto のレコード位置を内部表に登録する．続く処理で，以前暫定的に発生した参照文番号に修正が必要である（ソースプログラムに文番号が現れたとかその他の理由で）ことが判明した時点で，内部表中の該当文番号を修正し，修正されたことを示すフラグを立てておく．このようにして，1パスで FORTRAN コードを生成し終えた後，修正フラグが立っている箇所のみ，内部表中の位置情報に基づいて，直接，出力ファイルを書き換える．以上の手順をソースファイル中の各プログラム単位について行う．

以上の処理はファイル書換えを除いてすべて内部処理で実現できるために，ユーザから見た処理系の実質的な処理時間，つまり，“プリプロセス時間＋FORTRAN コンパイル時間” が従来の1パス処理に比して，遅くならないことが予想される．すなわち，数多く現れる冗長な continue 文，goto 文のファイル入出力を抑制することはプリプロセス，コンパイルの双方の時間短縮に有効であり，これは最適化出力を得るための上述のオーバヘッドを十分に補償すると考えられる．

3. プリプロセッサの概要

Westran の言語仕様については，文献2) に詳しいので，ここでは関連する1，2の事柄を除いて，説明は省略する．以下の説明では，とくに断わらない限り，continue 文，goto 文と記した場合，プリプロセッサが生成するそれらであるものとする．また，Westran ソースプログラム中の文番号をとくに “S文番号” と呼ぶことにする．プリプロセッサが発生する文番号は23000台を割り当てている．なお，基本的な留意点として，ソースプログラム中のS文番号，continue 文，goto 文はいっさい変更せず，変換後もそのまま出力することにした．

3.1 予約語の分類

goto 文の出力を抑制するかどうか，あるいは，内部表中の文番号を修正するかどうか等の重要な判断

表 1 Westran の予約語の分類
Table 1 Classification of Westran key words.

		最初が goto 文Ⓐ	最初が goto 文以外Ⓑ
最後が goto 文	次行に文番号を要求ⓐ	else, otherwise endwhile, endifor enddfor, pentry endp, enddoforever [goto ──]	perform [S1 goto ──]
	要求せずⓑ	exit, pexit [goto ──]	
最後が goto 文以外	(次行は要求しない文番号)ⓒ	elseif', when' [goto ── S1]	if', ifor', dfor' case', until while' [S1]
要求のみ文番号のみ	ⓓ	endif, endifor repeat, doforever	

・右肩に・印のついた予約語は前方参照を必要とする.
・[ ]内は予約語に対応して生成される FORTRAN コードであり, S1 は goto 文, continue 文以外の実行文（の集り）である.

は，直前に出力した文の種類と現在出力しようとしている文の種類の対応により決定される．そこで，Westran の予約語をそれに対応して生成される FORTRAN コードに基づいて分類した．その結果を**表1**に示す．同表中で pentry, pexit, pend, perform は，プログラム単位内でローカルな内部手続きの宣言と実行に関する予約語であり，Westran に特有の仕様である．

表1において，分類の例を挙げると，

・endwhile は繰り返すために，while の条件判定に戻るための goto 文と繰返しを脱出したときに制御が移ってくる文番号に変換され，たとえば，

```
        goto 23000
23001
```

といった FORTRAN コードが生成される．これを見ると最初の文は goto 文であるからⒶのタイプである．また，最後に出力される文も goto 文（この場合には最初の文と一致している）である．しかも次行に文番号23001を要求しているのでⓐのタイプでもある．したがって endwhile はⒶ—ⓐのタイプである．

・perform は，内部手続き実行後，直後に戻るために参照される変数に値を代入する文，内部手続きに飛ぶ goto 文，戻り先の文番号の三つに変換され，たとえば

```
        iname=1 (name は内部手続名)
        goto 23000
23001
```

が生成される．最初の文は goto 文でないのでⒷタイプ，最後の文は goto 文であり，次行に文番号を要求するのでⓐのタイプ，したがってⒷ—ⓐのタイプである．

・exit は繰返しを抜け出す goto 文に変換される．goto 文のみであるから，Ⓐ—ⓑタイプである．

・elseif は直前の文を実行後，if 構造の直後に飛ぶ goto 文と論理 if 文に変換されるのでⒶ—ⓒタイプである．

・endif は if 構造の直後に飛ぶ goto 文に対応する文番号が要求されるのみであるからⓓタイプである．

プリプロセッサが識別する FORTRAN 部分の予約語として次のものがある．

① プログラム単位の始まりを示す…
　block data, subroutine, function
② プログラム単位の終りを示す…end
③ 制御の流れを変える…goto, stop, return
④ その他…format

goto, stop, return 以外の FORTRAN 実行文は表1のⒷ—ⓒタイプに分類できる．これら以外に，とくにS文番号は，後に述べるように特定の処理を必要とするために，予約語の扱いをしている．

3.2 プリプロセッサが使用するデータ構造と変数

従来のプリプロセッサにおいては，構文解析用にスタックを用いており，読み込んだ予約語について，その種類とそれに対応して生成される文番号を格納している．ここでは，そのスタックは上記二つ以外にも新しい項目が追加され，拡張して使用される．さらに，このほかに，従来のプリプロセッサでは使用されなかった作業用のデータ構造と変数として，二つの表と二つの変数が使用されている．以下ではこれらについて説明する．**図2**にこれらデータ構造と変数の関係を示しておく．

(i) LABTBL

LABTBL は2章で述べた内部表のことであり，次の三つの情報が順次，格納される．①暫定参照文番号 SN，②後続する処理で，SN が修正を要すること

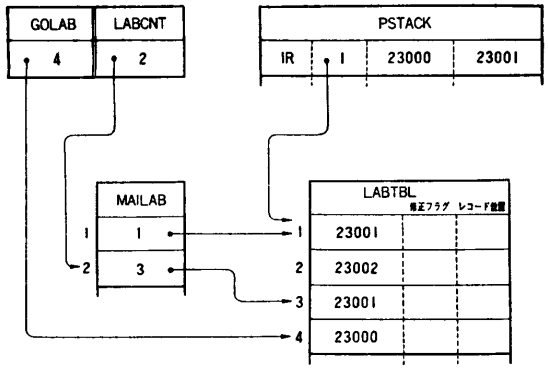


図 2 各種作業用データ構造の間の関係
Fig. 2 The relationship among data structures.

が判明した場合，修正した後，そのことをチェックしておくためのフラグ，③出力ファイル中で SN を含む行（レコード）の位置.

（ii） PSTACK

構文解析用のスタックであり，次の各情報を格納する．①予約語の種類を示す内部記号，②予約語に対応して生成される goto 文，if 文の暫定参照文番号について，それを格納している LABTBL 中の箇所を指示するポインタ（配列 LABTBL のインデクス），③個々の予約語に応じて，その役割が定まっている文番号（二つ以下），たとえば，ifor (incremental for) の場合は，繰返し用の文番号と繰返しを脱出するための文番号.

なお②のポインタの役割は次の(iii)や4章で触れる．図2では，IR (ifor) を続み込んだとき，繰返し脱出用の暫定参照文番号23001を発生し，まずLABTBL (1) に書き込んだ後，それへのポインタ1を PSTACK に格納している．また，たとえば ifor を脱出する exit 文を読み込んだとき PSTACK から IR を引き当てて，その脱出用文番号23001（4番目のフィールド）を LABTBL の次の空場所 (LABTBL (3)) に格納する.

（iii） MAILAB

次行に文番号を要求する予約語（表1におけるタイプⓐ,ⓓ）の場合，それに対応する予約語（たとえば，endif に対して if）を PSTACK から引き当て，要求されている文番号 SN をその3番目または4番目のフィールドに従って決定する．続いて2番目のフィールドに格納されている前述の LABTBL へのポインタで指される位置から LABTBL に最後に格納した位置の範囲内で SN を探し出す（一般に複数個ある）．MAILAB は一次元配列であり，その1番目の要素から，順次 LABTBL 中の該当箇所を指示するポインタが格納される．図2では endifor の読込みで LABTBL の1～4の範囲で23001が探し出され，その位置1，3が MAILAB に格納される．これらのポインタは次の入力語の種類によっては LABTBL 中の暫定参照文番号を修正する必要があるときに用いられる.

表 2 予約語のタイプに依存する二つの変数の値の遷移
Table 2 Updated values of two variables after the processing of each type of key words.

変数＼予約語のタイプ	ⓐ	ⓑ	ⓒ	ⓓ
GOLAB	>0	>0	=0	直前の値を保持
LABCNT	>0	=0	=0	>0

暫定参照文番号の発生，goto 文および論理 if 文の発生，さらに文番号の修正とチェック等，本プリプロセッサの最も重要な作業は，アルゴリズムの簡明化のために，有限オートマトンの状態遷移の考え方に基づいており，次の二つの変数により制御される.

（iv） GOLAB

直前の出力が goto 文であったとき，LABTBL 中のその参照文番号を指示するポインタが入る．直前の出力が goto 文以外のときは0とする.

（v） LABCNT

次行に文番号を要求する予約語の場合，その文番号を参照している goto 文，論理 if 文の数，したがってこの LABCNT は，(iii)の MAILAB に格納されているポインタの数を与えている．なお，次行に文番号が要求されないときは0とする.

図2の例は最後が goto 23000 で終わっており，次行に文番号23001を要求している例であり，この場合 GOLAB=4, LABCNT=2 となる.

Westran の予約語が入力されたとき，プリプロセッサは上記二つの変数により大略，次の手順で制御される.

① これら二つの変数の値と入力予約語が表1においてⒶ,Ⓑ,ⓓのどのタイプであるかに応じて必要な前処理を決める（この前処理については次節で述べる）.

② 前処理を実行後，入力予約語に対応して生成される FORTRAN コードをファイルに出力する.

③ 今度は同じ入力予約語がⓐ～ⓓのいずれのタイプであるかに応じて，これら二つの変数を表2に従って更新する.

表 3　各タイプの予約語と S 文番号の(前)処理
Table 3　(Pre-) processing of each type of key word and S-label.

EXGO (Ⓐタイプの前処理)	ⓓタイプの処理

if (LABCNT>0); then
　MAILAB を見て LABTBL 中の対応する文番号を修正してフラグを立てる
endif
if (GOLAB=0); then
　goto 文を出力する
else　　出力しない
endif
	if (LABCNT>0); then
	　MAILAB を見て LABTBL 中の対応する文番号を修正してフラグを立てる
	endif

NOGO (Ⓑタイプの前処理)	**WESLAB(S文番号の処理)**

if (LABCNT>0); then
　MAILAB を見て LABTBL 中の対応する文番号を出力する
endif
	TOK=次の入力語
	if (TOK が format でない); then
	　if (LABCNT>0); then
	　　MAILAB を見て LABTBL 中の対応する文番号を修正してフラグを立てる
	　endif
	　if (TOK が while か repeat か doforever); then
	　　LABCHK=S文番号
	　endif
	endif
	S文番号を出力する

3.3　予約語変換のための前処理ルーチン

すでに述べたように，入力予約語がⒶ, Ⓑ, ⓓのいずれのタイプであるかに応じて，必要な前処理が決定される．このほかに，S文番号が現れたときにも特定の処理が行われる．以下ではこれらについて説明する．

(i)　Ⓐタイプの予約語の前処理

以後，この前処理ルーチンを EXGO と呼び，その概略を表3に示す．EXGO ではまず LABCNT の値を調べる．LABCNT が0でなければ次行に文番号が要求されている．しかもⒶタイプであるから，たとえば次のような文番号付きの goto 文が作られることになる．

```
23000  goto 23001
```

この場合，もし 23000 を参照している箇所をすべて 23001 に書き換えたならば，要求されている文番号 23000 は出力する必要がなくなる．一方，23000 を参照している箇所は MAILAB 中に LABTBL へのポインタとして登録されているので，これらのポインタを使って LABTBL 中の該当箇所をすべて 23001 に修正してから修正フラグを立てておく．続いて GOLAB の値を調べる．GOLAB が0でない場合は，直前の出力が goto 文であるから，次行に goto 文を出力する必要はない．GOLAB が0のとき，すなわち直前の出力が goto 文でないときには，PSTACK から引き当てた文番号を参照文番号として goto 文を出力する．同時にその (暫定) 参照文番号および出力した goto 文の出力ファイルにおける位置を LABTBL に格納する．

(ii)　Ⓑタイプの予約語の前処理

この前処理ルーチンを NOGO と呼び，その概略を表3に示す．まず LABCNT の値を調べて0でない場合には，次行に文番号が要求されている．しかもⒷタイプであるから，最初の出力は goto 文ではない．したがって MAILAB 中のポインタを使って LABTBL 中の該当文番号を得，それをそのまま出力しておく．

ところで，直前の出力が goto 文であり，しかも LABCNT が0でないときに，たとえば次のような形になることがある．

```
        goto 23000
23000
```

この場合には goto 23000 は削除可能である．しかしこのようなパターンは滅多に現れないので，チェックのためのオーバヘッドとの兼合いを考えて，とくに考慮していない．

(iii)　ⓓタイプの予約語の処理

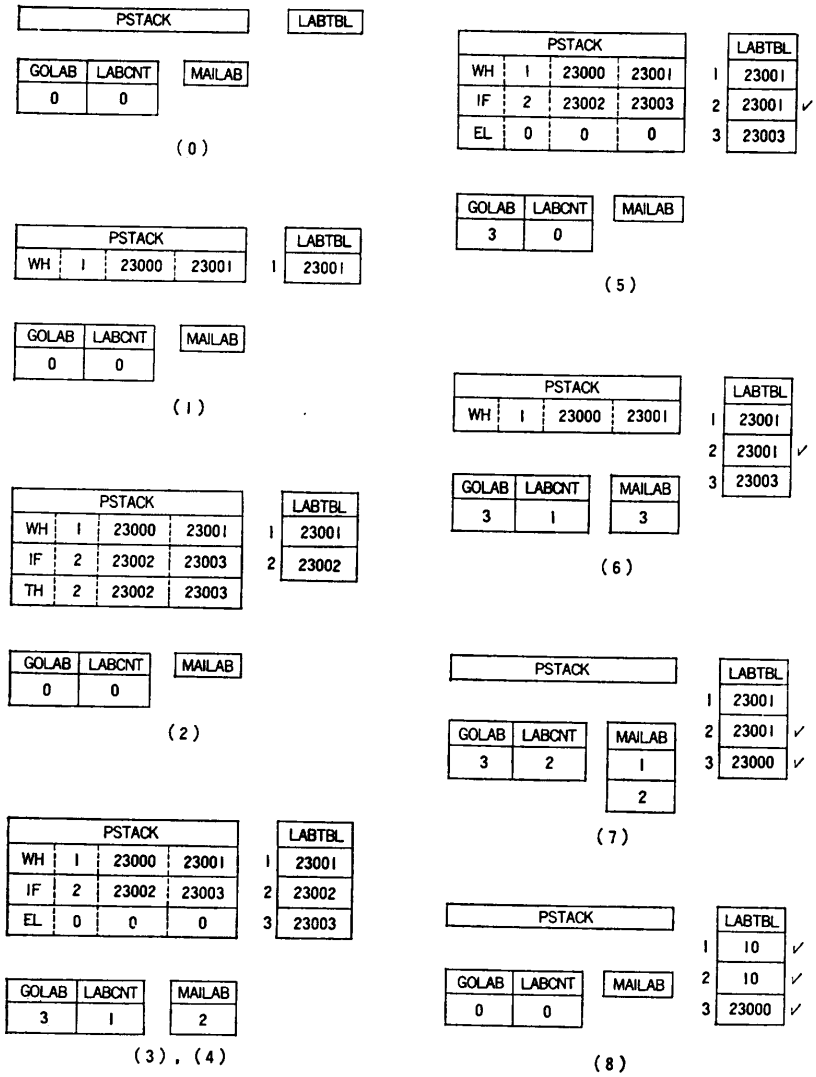
EXGO の前処理のうち前半部の処理を行う(表3)．

(iv)　S文番号の処理

このルーチンを WESLAB と呼び，その概略を表3に示す．まず，次の入力語 TOK を先読みする．TOK が format であれば，非実行文であるため goto の参照先とはなりえないのでそのままS文番号を出力する．format 文の文番号でないときには以下の処理を行う．LABCNT を調べて0でないときは次行に文番号が要求されている．しかし，この文番号はプリプロセッサが暫定的に発生した文番号ではなくて，現れたS文番号でなければならない．したがって MAILAB を参照することにより，対応している LABTBL 中の文番号をS文番号に修正して，修正フラグを立てておく．ただし，S文番号の次に，繰返し用の文番号を最初に要求する予約語である while, repeat, doforever が現れた場合は，変数 LABCHK にS文番号の値を入れておく．これらの予約語の処理に対してこのS文番号を引き継ぐために LABCHK を使って受け渡すためである．最後にS文番号を出力してから，GOLAB, LABCNT をともに0とする．

```
(1)      while( P1 )          →  23000 if(.not.( P1 )) goto 23001
(2)        if ( P2 ) ; then   →        if(.not.( P2 )) goto 23002
(3)          S1               →        S1
(4)        else               →        goto 23003
(5)          exit             →
(6)        endif              →
(7)      endwhile             →
(8)   10 S2                   →  10    S2
          (A)                                  (B)
```

図 3 パス1の直後に得られる変換出力
Fig. 3 The output obtained at the end of path-1.



✓印は修正フラグを表す．また LABTBL 中，レコード位置を表す情報は省略している．

図 4 図1(A)を変換する時使われるデータ構造のトレース
Fig. 4 Tracing of data structures in transforming Fig. 1(A).

3.4 Westran の予約語以外の処理

3.1 節で述べた予約語 stop, return, goto については，NOGO 処理を行う．つまり，直前に goto 文が出力されていても，これらの文はそのまま出力される（ソースプログラム中の goto 文はたとえ除去可能であっても保存する方針なので）．一方，これらの予約語を出力した後，次行にプリプロセッサが生成する goto 文を出力しようとしたときは，その出力は抑制される．さらに，いままで述べた以外の FORTRAN の実行文は 3.1 節で触れたように Ⓑ—Ⓒ タイプの分類に従って処理され，前処理として NOGO 処理が行われる．

3.5 出力ファイルの直接書換え

3.4 節までに述べた処理を1パスで行った後，LABTBL の内容に従って出力ファイルを直接修正する．すなわち，LABTBL 中の修正フラグが立っている箇所について，そのレコード位置情報と修正文番号に従って直接出力ファイルを修正して，最終的な出力を得る．

4. 構造化文の変換過程

3章において本プリプロセッサの動作を一般的に述べたが，ここでは，より具体的な説明として，構造化文（の入れ子）が，最適化された FORTRAN コードに変換される過程を，とくに冗長な文を抑制する過程を中心として説明する．紙面の都合上，1，2の例のみに留める．まず図1(A)の例について変換過程を説明する．**図3**(B)は図1(A)を1パス処理した直後の出力であり，入力行と対応づけて出力行を示してある．以下の説明の番号は図3(A)の入力行の番号に対応している．また，**図4**は各入力行を処理した直後の各データ構造と二つの変数 GOLAB, LABCNT のトレースである．なお図3(A)における S1, S2 は goto, stop, return 以

外の FORTRAN の実行文の集りであるとする．

（0） 初期値は GOLAB, LABCNT ともに 0 とする．

（1） while はⒷタイプ（最初が goto 文以外）であるから NOGO 処理を行うが，GOLAB. LABCNT ともに 0 により，何もしない．LABCNT＝0 より，引き継ぐべき文番号はないので，自ら繰返し用文番号 23000 を発生する．さらに繰返し脱出用の暫定参照文番号 23001 を発生し，LABTBL に格納する．同時に PSTACK に必要な四つの情報を整えて積み込む．また，while はⓒタイプであるから表2に従って GOLAB, LABCNT はともに 0 とする．

（2） if はⒷタイプであり NOGO 処理を行うが，GOLAB, LABCNT ともに 0 であるから何もしない．if の条件 P2 が成立しないとき，S1 をまたぎ越すための暫定参照文番号 23002 を発生し，LABTBL に格納する．P2 が成立するとき S1 を実行後 if 構造の直後に移るために発生した 23003 も含め，必要な情報を PSTACK に積み込む．また then の処理ではたんに，if のときに積み込んだ三つ情報をそのまま積み込む．if はⓒタイプなので，最後に表2に従って GOLAB, LABCNT ともに 0 とする．

（3） S1 は 3.1 節で述べたようにⓒタイプに分類されるので，S1 の処理後は GOLAB, LABCNT ともに表2に従って 0 とする．

（4） else の入力で PSTACK より TH(EN) を降ろし，S1 をまたぎ越すための文番号 23002 と if 構造の直後に移るための 23003 を得る．else はⒶタイプであり，EXGO 処理を行う．この処理では GOLAB＝0 より，goto 文が出力されるが，その参照文番号は先に得た 23003 であり，これは LABTBL にも格納される．また，else はⓐタイプであり，出力の最後が goto 文であるから，GOLAB には LABTBL へのポインタ 3 が入る．さらに，次行にさきほど得た 23002 が要求されるのでそれを LABTBL 中で捜し（3.2 節の(ⅲ)で述べた範囲内，つまり 2〜3 内で捜すが，この場合唯一つである），MAILAB にそれへのポインタ 2 を入れ，LABCNT にその個数 1 を入れる．

（5） PSTACK から最近の繰返し構造を見つける．この場合は WH であり，その脱出用文番号 23001 を得る．exit はⒶタイプであり，EXGO 処理を行うが，GOLAB＝3 より goto 文の出力は抑制される．一方，LABCNT＝1 であるから MAILAB を見て，LABTBL(2)に本来 exit で生成するはずの goto 文の参照文番号 23001 を書き込む．同時に修正フラグ（✓印）を立てる．LABCNT を 0 にする．GOLAB は goto 文の出力が抑制されるので 3 のままである．

（6） endif はⓓタイプであり次行に引き継ぐ文番号を決定する必要がある．まず PSTACK より EL と IF を降ろし，if 構造の直後に飛ぶ 23003 を得る．この文番号を LABTBL 中で捜し，それへのポインタ 3 を MAILAB に入れる．LABCNT を 1 とする．GOLAB は表2に従って 3 のままである．

（7） PSTACK から WH を降ろし，23000 と 23001 を得る．endwhile はⒶタイプであるから，EXGO 処理を行う．まず，LABCNT＝1 より LABTBL に修正を加える．つまり，本来 endwhile で出力すべき goto 文の参照文番号 23000 が LABTBL(3)に入る．また GOLAB＞0 より goto 文の出力は抑制される．一方，次行に文番号 23001 を要求する（ⓐタイプだから）ので 23001 を格納している LABTBL の箇所へのポインタ 1，2 を MAILAB に入れ，LABCNT＝2 とする．GOLAB は goto 文の出力が抑制されるので 3 のままである．

（8） 10 は S 文番号だから WESLAB 処理を行う．LABCNT＝2 より MAILAB を見て LABTBL の 1，2 番目を 10 に修正する．GOLAB, LABCNT ともに 0 とする．S2 の入力で NOGO 処理を行うが GOLAB, LABCNT ともに 0 であるから何もしない．

以上(1)〜(8)の 1 パス処理で，図3(B)がファイルに出力される．これに対して，LABTBL 中の修正フラグ箇所とレコード位置の情報に従い，出力ファイルを直接，修正し，正しい変換出力図1(D)を得る．

次に図1(A)において else がない場合（**図5**(A)）

```
   while( P1 )          →23000 if(.not.( P1 )) goto 23001     23000 if(.not.( P1 )) goto 10
     if ( P2 ) ; then   →       if(.not.( P2 )) goto 23002           if(.not.( P2 )) goto 23000
        S1              →       S1                                   S1
     endif              →
   endwhile             →       goto 23000                           goto 23000
10 S2                   →10     S2                           10      S2
        (A)                             (B)                                  (C)
```

図 5 図3(A)において“else”部分がない場合
Fig. 5 In the case of no “else” part in Fig. 3(A).

について説明する．図5(B)に1パス終了後の出力，図5(C)に出力ファイル修正後の最終出力を示す．(1)～(3)はまったく同様である．endif の読込みで，まず PSTACK から一つ降ろす．その記号が EL ではなく TH なので以下の処理を行う．if のときに出力した goto 23002 は else が以後に続くと仮定して出力したが，この場合は else がなかったのでこの goto は if 構造の直後に飛ぶことになる．つまり，TH を降ろしたときに得た23003が参照先となるので，同時に得た2番目のフィールドのポインタ2を使って LABTBL 中の23002を23003に修正してフラグを立てる．さらに，IF をおろしてから，LABTBL 中で23003を捜してそれへのポインタ2を MAILAB に入れる．LABCNT は1となり GOLAB は0のままである．続いて endwhile の読込みで，先の(7)と同様に PSTACK から WH を降ろし23000と23001を得て，EXGO 処理を行う．今度は，GOLAB=0だから goto 文を出力し，23000を LABTBL に格納する．また，LABCNT=1 より，LABTBL(2)が再度23000に修正される．

5. 本プリプロセッサの評価

二つの Westran ソースプログラム(A),(B)について，“従来のプリプロセッサ”を使用した場合と本プリプロセッサを使用した場合の比較を表4に示す．

表4 本プリプロセッサと最適化を行わないプリプロセッサの比較
Table 4 Comparison of this preprocessor with ordinary one which has no optimizing feature.

	ソースプログラム	(A)	(B)
従来のプリプロセッサ	全出力行数（行）	811	1,898
	うち，continue 文の数	216	345
	うち，goto 文の数	187	211
	プリプロセス時間 (sec)	20.33	66.04
	Fortran コンパイル時間	34.24	113.13
	合計時間	54.57	179.17
本プリプロセッサ	全出力行数（行）	543	1,489
	うち，修正フラグが立った行数	66	173
	うち，goto 文の数	138	161
	出力抑制された goto 文の数	49	50
	プリプロセス時間 (sec)	20.62	72.09
	うち，ファイル修正時間	1.90	6.03
	Fortran コンパイル時間	27.36	103.77
	合計時間	47.98	175.86

測定結果はいずれも YHP 1000 E 計算機におけるものである．

構造的 FORTRAN プリプロセッサの多くは文献1)に示されているプリプロセッサの作成法を採用していると考えられ，ここでいう“従来のプリプロセッサ”も“内部手続き構造”等，Westran 独自の仕様を除いてすべて文献1)の作成法に従って筆者らが作成したものである．また本プリプロセッサも最適化のための部分以外は文献1)に従って作成されており，プリプロセス速度を上げるための他の技法は使っていない．プログラム(A)は内部手続きを多用しており全体で一つのメインプログラムであり，実行文の割合が高い．したがって本プリプロセッサの最適化の効果が比較的，顕著に現れている．またプログラム(B)は多数のサブルーチンを含み宣言文の割合が高く，効果が現れにくい．

出力行数において20～30%の減少が確認される．このことは付録の(B)にも見られるように，文番号の減少とも相まって出力プログラムをかなり見やすくしている．この減少はまた，FORTRAN コンパイル時間を10～20%減少させている．一方，プリプロセス時間について，本プリプロセッサの時間から，1パス後ファイル修正に要する時間を除いた値が，従来のプリプロセッサの時間に比べて，若干減少しているかあるいはほぼ等しい．このことは，冗長な文を抑制するためのオーバヘッドが，出力行数の減少による時間減少で相殺されるのが原因と考えられる．結果的にユーザから見た全処理時間，“プリプロセス＋コンパイル”時間は本プリプロセッサのほうが速くなっている．また，変換後のプログラムの実行時間については，最適化プログラムのほうが確実に速くなるが，それはごくわずかである．なぜなら，continue 文はコンパイル後は，通常コードとしては現れず，また goto 文はジャンプ命令，唯一つのみに変換されるからである．したがって，大量の繰返しを行うループにおいて goto 文の出力が抑制される場合を除いて大して期待できない．本プリプロセッサは現在 YHP 1000 E FORTRAN Ⅳ および MELCOM COSMO 700 Ⅱ FORTRAN Ⅳ，拡張 FORTRAN の下で稼動している．

6. むすび

従来の構造的 FORTRAN プリプロセッサのほとんどが冗長な continue 文や goto 文を多数発生させるのに対して，本論文では基本的に1パスで最適化出力を生成するプリプロセッサの方式を新しく提案し，それに基づいてプリプロセッサを作成し，種々の比較・

評価を行った．その結果本プリプロセッサの有効性を確認した．本方式が適用できるための条件として，変換出力ファイルは修正時になんらかの方法で直接アクセス可能でなければならない．YHP の FORTRAN Ⅳでは逐次ファイル中の任意のレコードに対してその位置を与えたり，任意の位置にファイルポインタをセットしたりするためのライブラリルーチンを使用している．また MELCOM ではキー付きファイルのキーによる直接アクセス機能を使用している．

最後に，今後の課題として，本方式を構造的 FORTRAN 言語として，より汎用性のある Ratfor に適用することが挙げられる．

謝辞　本研究の一部は，文部省科学研究費補助金の援助を受けた．

参考文献

1) Kernighan, B. W. and Plauger, P. J.: *Software Tools*, Addison-Wesley Publishing Company, Reading, Mass. (1976).
2) 真野，杉藤，鳥居：Westran：Fortran をベースとした構造的言語とその処理系，情報処理，Vol. 18, No. 8, pp. 808-813 (1977).
3) Guida, G.: An Effective Preprocessor for Structured FORTRAN: The HENTRAN System, *Int. J. Comput. Inf. Sci.*, Vol. 10, No. 4, pp. 283-297 (1981).
4) 石田晴久：UNIX，第 15 章，共立出版，東京 (1983).
5) 都司，岡嶋，渡辺：構造的 FORTRAN プリプロセッサ出力における CONTINUE 文と GOTO 文の除去，電子通信学会論文誌 (D)，Vol. J 67-D, No. 4, pp. 544-545 (1984).

付　録

(A)　Westran ソースプログラム

```
0001          doforever
0002            perform INLNE
0003            if (ISN.eq.23); then
0004              if (SK.eq.2 Hco); then
0005                IC=IC+1; TEMP (IC)=ISN
0006                while (M.lt.IC)
0007                  TABLA (M, L)=TEMP (M)
0008                  M=M+1
0009                endwhile
0010                else
0011                UNGET=.true.; exit
0012              endif
0013              elseif (SK.eq.2 Hgo); then
0014                UNGET=.true.
0015                ifor K=1, IC
0016                  TABLB (K, L)=TEMP (K)
0017                endifor
0018            endif
0019          enddoforever
0020  100     MOVLAB=.false.
```

(B)　(A)に対する本プリプロセッサ出力

```
0001  23000   IINLNE=1
0002          GOTO 23002
0003  23003   IF (.NOT.(ISN.EQ.23)) GOTO 23004
0004          IF (.NOT.(SK.EQ.2 HCO)) GOTO 23006
0005          IC=IC+1
0006          TEMP (IC)=ISN
0007  23008   IF (.NOT.(M.LT.IC)) GOTO 23000
0008          TABLA (M, L)=TEMP (M)
0009          M=M+1
0010          GOTO 23008
0011  23006   UNGET=.TRUE.
0012          GOTO 100
0013  23004   IF (.NOT.(SK.EQ.2 HGO)) GOTO 23000
0014          UNGET=.TRUE.
0015          K=1
0016          I 23011=IC
0017          GOTO 23011
0018  23012   K=K+1
0019  23011   IF (K.GT.I 23011) GOTO 23000
0020          TABLB (K, L)=TEMP (K)
0021          GOTO 23012
0022  100     MOVLAB=.FALSE.
```

(C)　(A)に対する従来のプリプロセッサ出力

```
0001  23000   CONTINUE
0002          IINLNE=1
0003          GOTO 23003
0004  23002   CONTINUE
0005          IF (.NOT.(ISN.EQ.23)) GOTO 23004
0006          IF (.NOT.(SK.EQ.2 HCO)) GOTO 23006
0007          IC=IC+1
0008          TEMP (IC)=ISN
0009  23008   IF (.NOT.(M.LT.IC)) GOTO 23009
0010          TABLA (M, L)=TEMP (M)
0011          M=M+1
0012          GOTO 23008
0013  23009   CONTINUE
0014          GOTO 23007
0015  23006   CONTINUE
0016          UNGET=.TRUE.
0017          GOTO 23001
0018  23007   CONTINUE
0019          GOTO 23005
0020  23004   IF (.NOT.(SK.EQ.2 HGO)) GOTO 23010
0021          UNGET=.TRUE.
0022          K=1
0023          I 23011=IC
0024          GOTO 23011
0025  23012   K=K+1
0026  23011   IF (K.GT.I 23011) GOTO 23013
0027          TABLB (K, L)=TEMP (K)
0028          GOTO 23012
0029  23013   CONTINUE
0030  23010   CONTINUE
0031  23005   CONTINUE
0032          GOTO 23000
0033  23001   CONTINUE
0034  100     MOVLAB=.FALSE.
```

(昭和59年4月 9 日受付)
(昭和59年7月19日採録)