U.D.C. 681.326.06 : [371.3 : 681.322.06 : 800.92 FORTRAN]

電子計算機教育用FORTRANコンパイラ“FORTCG”の開発

Development of Compile-and-Go FORTRAN Compiler

渡辺道生* *Michio Watanabe*
加藤正道* *Masamichi Katô*
中所武司* *Takeshi Chûsho*
林　英治** *Eiji Hayashi*

A high speed Compile-and-Go FORTRAN compiler (FORTCG) has been developed for use with Hitachi Computer 800 series EDOS and EDOS-MSO systems. The FORTCG is intended for both instruction in FORTRAN programming and debugging of FORTRAN programs. Its language specifications are the same as the FORTRAN IV, but it also contains statements for debugging. The FORTCG has achieved high speed processing capability by the 1-pass compiling and other techniques. It is also provided with a mechanism capable of extensive error checking during compilation and execution.

1 緒　言

一般にあるプログラムを，例えば，FORTRAN言語で書きその結果を得るには，そのプログラムをFORTRANコンパイラ及びリンケージ エディタで処理した後，実行しなければならない。ここで，通常のFORTRANコンパイラは，オブジェクト プログラムの効率，他言語で書かれたプログラムとの結合，比較的大きなプログラムの処理の可能性などに重点が置かれ汎用になっているため，大がかりで，かつコンパイル時間がかかり，使い方も複雑である。その他，リンケージ エディタやコントロール プログラムが汎用になっているため，更に処理時間がかかる。

一方，学校などの電子計算機教育におけるように，プログラム エラーを含むことが多く，コンパイルの回数に対し，そのオブジェクト プログラムを実行する回数が少なく，また，比較的小さなプログラムを多量に処理する必要がある場合には，コンパイル時間及び全体のスループット時間の短いことが要求され，通常のFORTRANコンパイラは，このような用途には適していない[1]~[3]。

今回，HITAC 8000シリーズの主力システムである Extended Disk Operating System（以下，EDOSと略す）及び Extended Disk Operating System-Multi Stage Operation（以下，EDOS-MSOと略す）システムに対して，コンパイル時間及び全体のスループット時間の短い Compile-and-Go FORTRANコンパイラ（以下，“FORTCG”と略す）を開発した[4]。この“FORTCG”は，前述した電子計算機教育用のほかに，FORTRANプログラムのデバッグ用としても有用であるよう設計した。以下に，“FORTCG”の機能，特長などにつき述べる。

2 “FORTCG”開発の方針

“FORTCG”は，電子計算機教育用及びデバッグ用を目的とし，次のような機能を持つFORTRANコンパイラを開発することにした。すなわち，

(1) コンパイル及び全体のスループットを速くし，その他，電子計算機教育用FORTRANに有用な機能（例えば，各文の使用頻度や犯したエラーの頻度など，プログラミングの習得度を知るのに役立つ各種の統計データをとる機能など）を備える。

(2) 言語仕様は，JIS FORTRAN 7000レベルを含み，一般にFORTRAN Ⅳと呼ばれているもの（例えば，EDOS-MSO FORTRAN Ⅳ）と同等の仕様とする。

(3)エラー チェック，特に通常のコンパイラでは，実行効率が低下するという理由で行なわれない実行時の動的なエラー チェックを強化し，更に，デバッグのための文を追加し，FORTRANプログラムのデバッグをしやすくする。

3 “FORTCG”の特長

“FORTCG”は，通常のFORTRANコンパイラにない，いくつかの特長を持っている。この特長を，言語仕様，機能仕様，性能とに分けて以下に述べる。

3.1 言語仕様

電子計算機教育用のためのFORTRANの言語仕様としては，ある機種に特有な仕様ではなく，標準的なものが望ましい。現在，FORTRANの言語の標準として，日本工業規格JIS FORTRANがある。そのJIS FORTRANにも，三つのレベルがあるが，“FORTCG”の場合，開発するシステムの規模から見て，その仕様のいちばん大きなレベルの7000に合わせ，これを含むことにした。

デバッグ用という面から見ると，デバッグ用コンパイラの出力するオブジェクト プログラムは，一般的に種々のチェックが入り，最適化も行なわれないので実行効率は悪い。したがって，デバック用コンパイラを用いて，デバッグ終了したプログラムは，その後，何回もプロダクション ランを行なうのであれば，プロダクション用のコンパイラで再びコンパイルして，効率の良いオブジェクト プログラムを作成したほうがよい。このとき，デバッグ用コンパイラの言語仕様とプロダクション用コンパイラの言語仕様が一致していることが望ましい。“FORTCG”の場合，開発するEDOS-MSOシステムのFORTRAN Ⅳと同等の仕様とした。ただし，「宣言

* 日立製作所システム開発研究所　** 日立製作所ソフトウェア工場

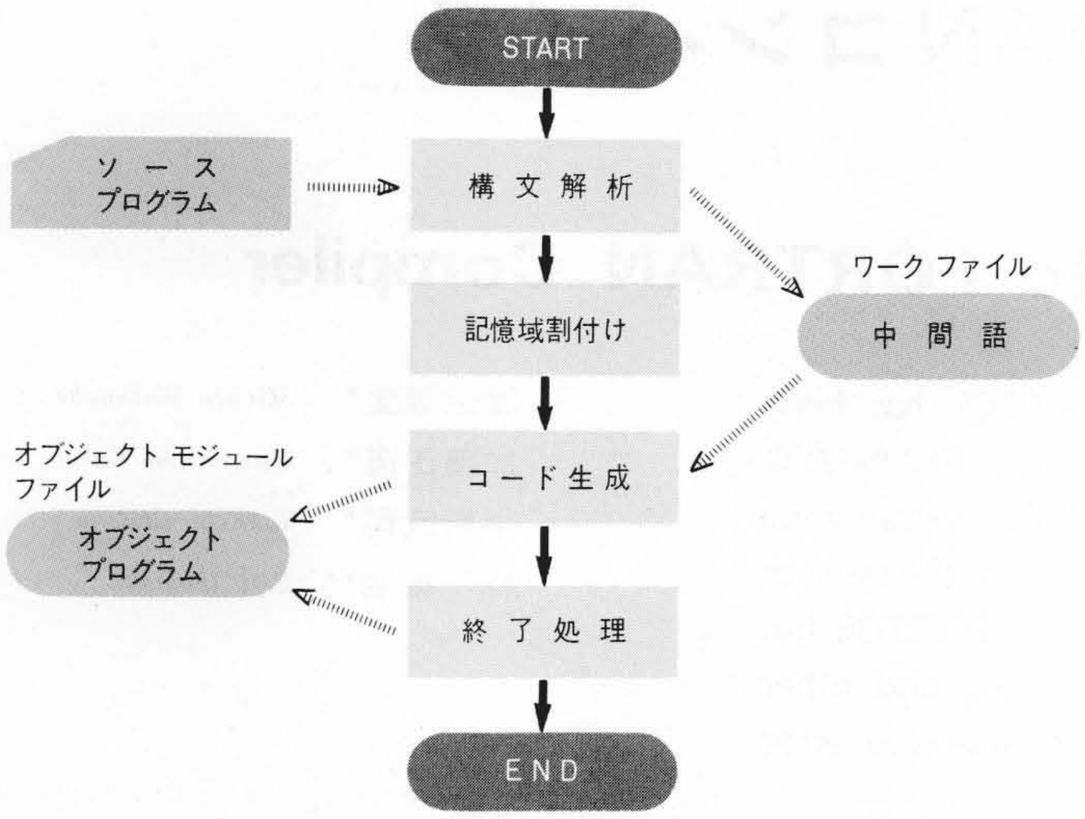

図1　通常のFORTRANコンパイラにおける処理の流れ　通常のFORTRANコンパイラでは，マルチ パスになっていて，その間にワーク ファイルとの入出力が行なわれる。

Fig. 1　Flow of Process in a Conventional FORTRAN Compiler

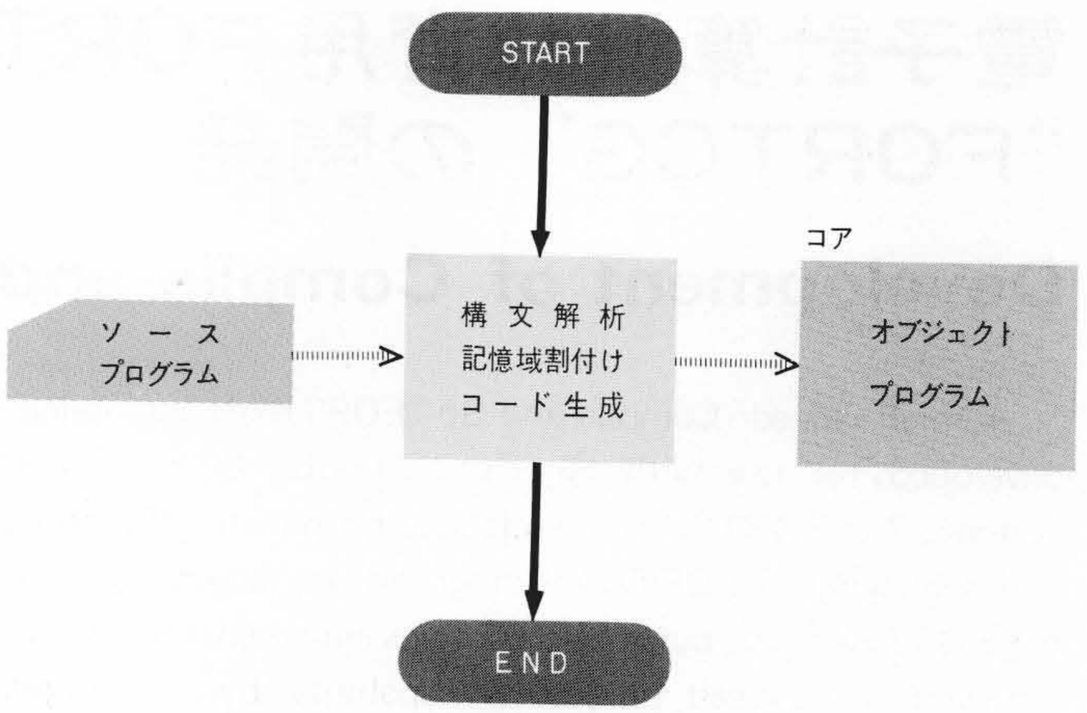

図2　“FORTCG”における処理の流れ　“FORTCG”では1パスでワーク ファイルとの入出力も行なわれず，オブジェクト プログラムは直接実行可能な形でコア上に作成される。

Fig. 2　Flow of Process in the FORTCG

文は，実行文の前に置くこと」という制限を付けた。この制限は，後述するが，“FORTCG”が1パス コンパイル方式をとっているためによるものである。

また，デバッグ用として，FORTRANプログラムを実行したときの，制御の流れ，変数の値の変化などの情報を出力するよう指定のできるデバッグのための文を追加した。

3.2　機能仕様

“FORTCG”の機能を，コンパイラの高速化のための機能，デバッグを容易にするための機能，教育用機能の三つに分類して以下に述べる。

3.2.1　コンパイラの高速化のための機能

学校などでプログラムの実習を行なうとき，大勢の学生のプログラムを多量に処理しなければならない。しかし，それらの学生のプログラムは比較的短く，エラーのある可能性が大きい。また，そのコンパイルした結果のオブジェクト プログラムは，1回だけ実行されるのがほとんどである。

したがって，教育用コンパイラとしては，オブジェクト プログラムの効率よりもコンパイルの速度が重視され，コンパイル及び実行を含めた全体の時間の短いことが要求される。

デバッグ用についても，コンパイル速度の速いことが望ましい。

“FORTCG”では，速度を速めるために以下の機能を持っている。

(1)　1パス コンパイル機能

通常のFORTRANコンパイラは，**図1**に示すようにいくつかのパスを経て，途中，コンパイルの中間結果である中間語をワーク ファイルと入出力を行ない，最終的にオブジェクト モジュール ファイルを作成する。これに対し，“FORTCG”では，**図2**のように1パスでソース プログラムから，そのまま実行できる形のオブジェクト プログラムをコア上に作成し，コンパイル途中にワーク ファイルを使用しない。これによって，コンパイラがオーバ レイになっているための各パスのロード，オブジェクト プログラムのロード，コンパイル途中のワーク ファイルとの入出力，マルチ パスになっているための処理のオーバ ヘッドなどの時間をなくすことができ，高速コンパイルを可能にしている。

(2)　プログラムの連続処理機能

通常のシステムでは，プログラムの実行を行なう場合，コンパイラの出力したオブジェクト プログラムをリンケージ エディタで処理する必要があるが，この時間及びいくつかのプログラムを連続処理するときのコントロール プログラムの時間は，処理すべきプログラムが小さいとき，コンパイル及

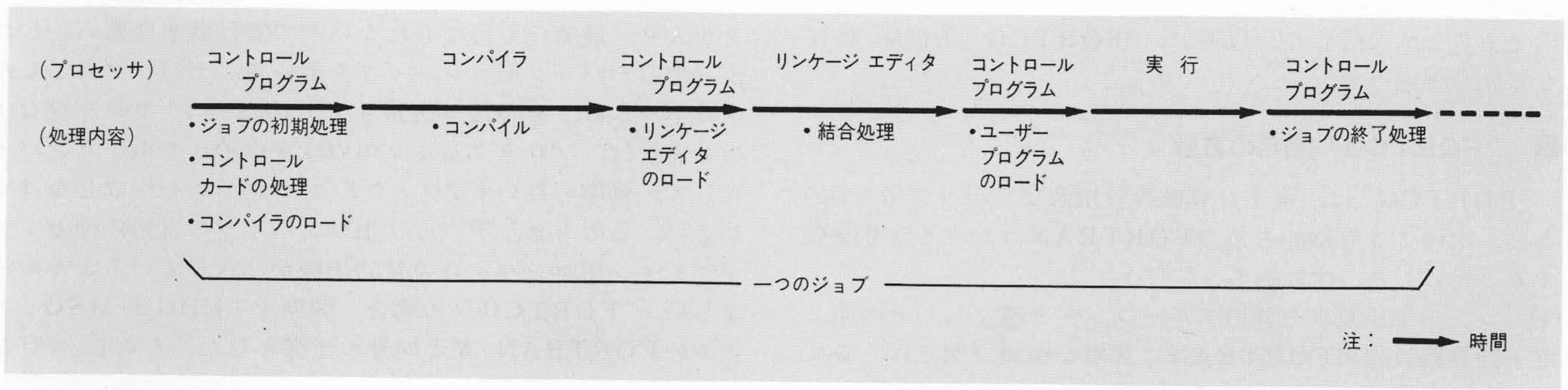

図3　通常のシステムにおける処理の時間経過　通常のシステムでは，コントロール プログラム，リンケージ エディタのオーバ ヘッドが大きい。

Fig. 3　Time Flow of Processing in a Conventional System

び実行の時間に対して大きな割合を占めることがある。**図3**は，これを示すものである。“FORTCG”では，**図4**のようにFORTRANプログラムのみからなる場合のコンパイル アンドゴーに焦点を合わせ，次の機能によりこれらを高速に処理できるようにした。

(a)　“FORTCG”独自のコントロール カードを使用して，コントロール プログラムを経由しないで，いくつかのプログラムの連続処理ができる。**図5**はその例を示すものである。

(b)　プログラムが，メイン プログラムといくつかのサブ プログラムから成るとき，これらの結合処理をコンパイラの中で行なう。

3.2.2　デバッグを容易にするための機能

初心者のプログラムはエラーを含んでいることが多く，また，初心者にとってエラーの場所や原因を見つけるのは難しいものである。教育用及びデバック用コンパイラとしては，エラーを発見しやすい機能が要求される。

エラーには，大別して，文法的なエラーと論理的なエラーの2種類がある。文法的なエラーは，言語仕様に違反したプログラムを書いたときのもので，その多くはコンパイラによって検出される。しかし，文法的なエラーの中でも，二つ以上のコンパイル単位の相互関係に依存するものや実行結果に依存するものは，通常のコンパイラでは，オブジェクト プログラムの効率が落ちるという理由で検出されないことが多い。この種のエラーの例としては，外部手続きを呼び出すときの仮引数と実引数の組合せの妥当性や，添字付き変数の添字が変数や式で指定されたときの添字の値の妥当性などがある。

論理的なエラーは，文法的には正しいが，プログラムが期待したように動かない場合をいう。プログラムの実行結果が予期したものと違うとき，論理的エラーがあることは分かるが，どこが正しくないかを見つけるのは一般的に困難である。論理的エラーを見つけるには，計算過程を遂次追跡すればよいが，このために，制御やデータの流れを情報として出力することを指定するデバッグのための文が便利である。

“FORTCG”では，デバッグを容易にするために，次の機能を持っている。

(1)　デバッグのための文を言語仕様として追加した（このことは前述した）。

(2)　コンパイル時の文法的エラーに対するチェックは，なるべく，詳しく行なうことにした。実行時のエラーも含めて，エラー メッセージの種類は約300個である。

(3)　通常のコンパイラでチェックされない以下の項目についてチェックを行なうことにした。これらの項目は，初心者が

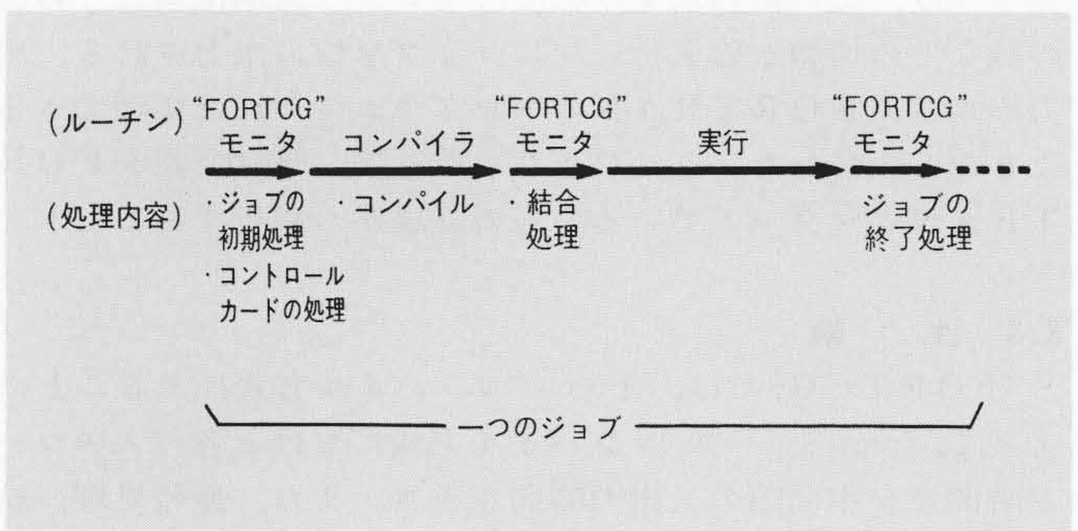

図4　“FORTCG”における処理の時間経過　　“FORTCG”では，連続処理，結合処理をコンパイラの中のコントロール ルーチンで行なう。

Fig. 4　Time Flow of Processing in the FORTCG

```
//    JOB                          } EDOSあるいはEDOS-MSOの
//    EXEC FORTCG                  } コントロールカード
@ JOB                              ----- “FORTCG”ジョブの開始を示す。
@ FORTRAN                          ----- ユーザー ジョブの開始を示す。
    ソース プログラム              ----- 一つのメイン プログラムといくつかの
                                         サブプログラム
@ EXEC
    データ                         ----- 実行時に読み込むデータ
@ FORTRAN                          ----- 次のユーザー ジョブの開始を示す。
    ソース プログラム
@ PARAM                            ----- コンパイラ オプションを変更する。
    ソース プログラム              ----- いくつかのサブプログラム
@ EXEC                             ----- 入力データがないとき。
@ FORTRAN
    ソース プログラム
@ STARTC  SUB1                     ----- ソース ライブラリからSUB1という
                                         サブ プログラムを取り込む。
@ EXEC
    データ
@ END                              ----- “FORTCG”ジョブの終了を示す。
```

図5　“FORTCG”の連続処理の例　　第1欄が@のカードを“FORTCG”のコントロール カードとして，いくつかのユーザー ジョブを連続処理する。

Fig. 5　An Example of Sequential Processing in the FORTCG

比較的犯しやすいエラーであり，また，見つけにくいものである。

(a)　飛び先のチェック

計算形ＧＯＴＯ文及び割当て形ＧＯＴＯ文の変数の値をチェックして，変なところへ飛び込んで暴走しないようにしている。エラーの例は，**図6**に示すとおりである。

(b)　添字のチェック

添字付き変数の添字の値が，宣言された配列の範囲内に入っているかどうかチェックする。添字の値のエラーは，添字が変数や式で指定された場合によく起こるもので，そのような添字付き変数が代入文の左辺に用いられると，配列の範囲外の場所がこわされることになり，原因不明のエラーが起こることもある。エラーの例は，**図7**に示すとおりである。

(c)　引数のチェック

外部手続きを引用するとき，実引数と仮引数の順序，数，種類及び型と長さが一致するかどうかチェックする。このエラーは，プログラム単位の異なったものの間の情報の受け渡しであり，比較的犯しやすく，また，見つけにくいも

```
      :
I=4
      :
GOTO(20,30,40),I
      :
          (1)

      :
ASSIGN 20 TO J
      :
GOTO J,(30,40,50)
      :
          (2)
```

図6　飛び先の誤り例　　計算形GOTO文，割当て形GOTO文でIあるいはJがそれぞれ範囲内に入っていない。いずれの場合も次の文が飛び先となる。

Fig. 6　An Example of Error in GOTO Statements

```
      :
DO 10 I=1,5
      :
CALL SUB(A,I)
      :
10 CONTINUE
      :
END                          ┐ メイン プログラム
SUBROUTINE SUB(A1,I1)        ┐
      :                      │
I1=2                         │ サブルーチン
      :                      │
RETURN                       │
END                          ┘
```

図9　DOパラメータを変更している例　　DOループの制御変数がサブルーチンSUBの中で変更されている。

Fig. 9　An Example of Error in DO Parameters

のである。エラーの例は，**図8**に示すとおりである。

(d)　未定義変数のチェック

変数に値を代入しないで，引用した場合のチェックを行なう。このエラーは，論理的エラーに属するものであり，実行するたびに結果が違う可能性があり，発見の困難なものの一つである。

```
DIMENSION A(3,4)
      :
I=4
      :
A(I,4)=1.0
      :
```

図7　添字の誤り例　　Aを引用するときは，A(3, 4)の範囲内に入っていなければならない。

Fig. 7　An Example of Error in Subscripts

```
REAL*4 A
      :
CALL SUB(1.0,A)              ┐ メイン プログラム
      :                      │
END                          ┘
SUBROUTINE SUB(I,A1)         ┐
REAL*8 A1                    │ サブルーチン
      :                      │
                             ┘
```

図8　引数が一致しない例　　実引数と仮引数とは，個数，順序，種類，形及び長さが一致しなければならない。

Fig. 8　An Example of Error in Arguments

例えば，

$A=B$という代入文の変数Bには，この文を実行する以前に，代入文，ＤＡＴＡ文，ＲＥＡＤ文などで，値がセットされていなければならない。

(e)　ＤＯの制御変数のチェック

ＤＯの制御変数がＤＯループの実行中に変えられていないかどうかチェックする。これは，ＤＯループが，ＤＯの拡張範囲を持つ場合や，**図9**のようにＤＯループの中から外部手続きを引用して，制御変数をその実引数としている場合，ＤＯの制御変数が変更される可能性があり，もし変更された場合，無限ループになる可能性があるためである。

3.2.3　教育用機能

教育用機能としては，プログラムに関する学生の習得度や傾向の情報が必要である。これらのデータとしては，どういう文をどれぐらい使用しているか，どういうエラーを犯しやすいかなどが考えられる。

“ＦＯＲＴＣＧ”では，各ユーザー ジョブごとにアカウント情報及び統計情報を出力する機能を持っている。

アカウント情報は，使用時間，印刷ページ数，使用メモリ量，終了原因などを印刷出力するもので，料金計算のためだけでなく，そのジョブのコンパイル及び実行について得られるいろいろの情報をユーザーに知らせる役目も持っている。

統計情報は，アカウント情報のほかに，エラーの番号別の発生頻度，文の種類別使用頻度，プログラム単位当たりの文の数などの情報が磁気テープにジョブ単位に出力される。出力形式は，ＦＯＲＴＲＡＮのプログラムでそれを処理できるようなものにした。これにより，教師は，統計情報をＦＯＲＴＲＡＮプログラムでいろいろの立場から解析することができる。

3.3　性　　能

“ＦＯＲＴＣＧ”では，１パス コンパイル方式にすることによって，マルチ パス コンパイル方式における各パスのロード時間及び中間語の入出力時間を省き，また，連続処理，結合処理をコンパイラの中で行なうことにより，FORTRANプログラムのコンパイル アンド ゴーのときのコントロールプログラム及びリンケージ エディタのオーバ ヘッドをなく

し，処理の高速化を図っている。

一方，コンパイル速度を速めるため，オブジェクト プログラムの最適化を行なっていないこと，コンパイル時及び実行時の各種のチェックを強化したことなどによる処理速度の低下がある。

“FORTCG”の処理速度を実測した結果，STOP文とEND行のみからなるプログラムで，実行を含めた全体のスループット時間は，4·7秒である。ただし，この測定条件は，

機種：HITAC 8450
システム：EDOS-MSO バージョン04
入力：ディスク内の入力スタックから入力
出力：ライン プリンタへ直接印刷出力
メモリ：“FORTCG”のために100,000バイト使用
測定方法：10個のSTOP-ENDジョブのコンパイル アンド ゴーを連続処理し，その1/10を一つのSTOP-ENDジョブの時間とした。

である。

一方，同一システムのマルチ パス コンパイラの同一条件におけるスループット時間は，46.2秒であり，“FORTCG”は，STOP-ENDジョブについて約10倍の処理速度を達成できた。

一般のプログラムの処理速度については，そのプログラムの内容に依存するが，一例として，1から30までの階乗を計算するプログラム（文の数45）のコンパイル アンド ゴーの場合，マルチ パス コンパイラで55秒に対し，“FORTCG”で11秒と5倍の処理速度向上の結果を得ている（ただし，この場合の電子計算機は，HITAC 8350である）。

4 結　言

HITAC 8000シリーズEDOS及びEDOS-MSOシステムに対し，電子計算機教育用及びデバッグ用として開発された“FORTCG”の開発方針と言語仕様及び機能の特長につき述べた。

電子計算機教育用及びデバッグ用から要求される言語仕様並びにその他の機能を検討し，これらのほとんどを実現することができた。今後，学校などで活用いただき，結果について大方の御批判を仰ぎたい。

参考文献

(1) Peter W. Chantz ほか：“WATFOR-The University of Waterloo FORTRAN Ⅳ Compiler”. CACM vol.10 No.1 (Jan.1967)
(2) D.D. Cowan ほか：“Design Characteristics of the WATFOR Compiler.” SIGPLAN Notices (July 1970)
(3) Saul Rosen ほか：“PUFFT-The Purdue University Fast FORTRAN Translator.” CACM vol.8 No.11 (Nov. 1965)
(4) プログラム マニュアル：EDOS/EDOS-MSO Compile and Go FORTRAN プログラミング文法 (8000-3-015) 日立製作所(昭和49年2月)
“FORTCG”の言語仕様，使用法について詳細に述べている。

論文抄録

LR(k)-Parser生成システムとそのFORTRANコンパイラへの応用

日立製作所　小島富彦・加藤正道，他1名

情報処理　15—2，93（昭49-2）

ソフトウェアの生産性向上は，最近ますます重要な課題となってきており，特にコンパイラについては多くの努力がなされてきた。我々は，言語理論を利用してプログラミング言語の文法から，LR(k) Parser（構文解析プログラム）を生成するプログラム，LR(k) アナライザを開発し，これを用いてFORTRANのParserを作り，それを基にして超小形計算機HITAC 10でFORTRANコンパイラを作成した。

従来，自動作成したParserは，構文解析アルゴリズムが汎用的であるために処理速度が低下するということと，プログラム サイズが大きくなるという問題とがあった。これを解決するため，我々はParserの最適化を自動的に行なわせるアルゴリズムを考案し，それをLR(k) アナライザに組み込んで処理速度を向上させるようにした。一方，Parserをコンパクトな「表」の形で表現することにより，プログラムの小形化を図った。

LR(k)アナライザでParserを完成させるには，まず，プログラミング言語の文法をバッカス記法で記述する。次に各ステートメントのオブジェクト コードを設計して構文規則の，対応する部分に書き加える。更に意味処理を考慮して，必要な処理ルーチン名を構文規則中に挿入する。

このようにして得られた文法を入力すると，LR(k)アナライザは，この文法で定義される言語を認識する決定性プッシュダウン オートマトンを構成し，それを構文解析表の形にまとめParserを作り上げる。

この段階でLR(k) アナライザは，次のような最適化をParserに施す。

(1) オートマトンの状態遷移系において，一つの状態から出ている遷移群と，別の状態から出ている遷移群との間の包含関係を見つけて，遷移表の共有を図る。
(2) 状態スタックの一番上に載っている状態名のチェックを，遷移表を小さくすることのできるような順序で行なわせる。
(3) 句を生成する状態のうちで，意味処理を伴わない状態を消去することにより，オートマトンを簡略化する。

更に，実際にコンパイラを作成するときには，これに加えて人手による最適化も行なう。そのため我々は，ソース プログラムを中間語に変換しながらParserの動作解析を行なうシミュレータを用意した。これを用いて，人手で最適化したあとのParserが正しく動作することを確かめたり，意味処理を実行する時点を把握したりする。

HITAC 10のコア4k語用（1語16ビット）の常駐FORTRANコンパイラの場合，コンパイラのサイズを小さくすることが技術上の重要な問題であった。LR(k) アナライザを使用して，ほぼJIS水準 3000の言語仕様に対するParserを280語の表にまとめ，これに表のインタプリタ，意味処理，入出力ルーチン，ライブラリ関数などを加えて，1パスの常駐コンパイラを，1.4k語で実現することができた。