Automatic Coding について (IV)

—— FORTRAN Compiler (2) ——

穂 鷹 良 介

目　　次

4. PHASE II

4. 1. 概 要

この PHASE では補助記憶装置に入っている statement の分類結果を使用して statement を後に定めるような規則に従ってシラブルに分解し，その結果を再び補助記憶装置に格納するのが目的である。

シラブルとここでいうものは一つの statement をさらに分解して得た意味のある最小の情報の単位であって，例えば

10 X(10)＝ABC＋B

という statement は 10, X, (, 10,), =, ABC, +, B, (end) ((end) は statement の終了を示す end 記号) という 10 個のシラブルに分解される。

この分解されたシラブルを使うのは PHASE III であるから，PHASE III

に持越す必要のない情報，例えばnameの文字の配列そのもの等はこのシラブルに盛り込まれない。

その代り，変数のシラブルには，割付けられたaddressの情報，整数型または実数型の区別等が盛られる。

このため，シラブル分解を行なう前に，配列，COMMON，EQUIVALENCEの処理を前以って行なっておかねばならない。また，3.でも述べたように

$$F(I, J)=I+J$$

等という statement は F という名前が配列名の中にあれば添字の計算をする数式であり，配列名の中になければ，statement 関数と考えられるから，この分類を行なうためにも，はじめに配列について処理を行なう必要がある。

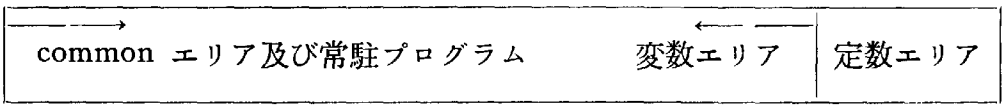
4．2．宣言文，メモリの割付

＜宣言文の取扱い＞

JIS 3000 で考えねばならぬ宣言文は DIMENSION 文，COMMON 文，EQUIVALENCE 文の三つである。JIS 3000 では，これらの宣言文は，上の順序で現われねばならぬとしているが，ここではその制限をとり去った場合を考えよう。

PASS の回数が1回不足なため，宣言文は実行文の前に現われ，また最初のプログラムの common block が最大であるという制限を設ける。

common block を含めて変数の割付エリアは次のようになっている。



宣言文はすべて実行文の前に現われることにしたから，一番目のプログラムの実行文がはじめて現われた所で common エリアが確定する。COMMON 宣言に現われた変数または配列，また EQUIVALENCE 宣言によってそれらと結合された変数または配列を common エリアに割付ける。それ以外の

変数または配列は変数エリアに割付ける。定数は固定したエリアを適当にとって割付ける。この際，定数エリアの使い残し，あるいは使いすぎが，他のエリアと独立に生ずるが，これはもう1回PASSの回数をふやさないと解決されない問題であろう。

これらのことを行なうために，一口でいうと次のようなcompile方法をとる。

実際の番地割付はすべての宣言文が現われた時に行ない，それまでは浮動の状態にしておく。つまり，COMMON, EQUIVALENCE 等の変数の相対的な結合関係だけを table に残しておくようにする。EQUIVALENCE 宣言に出て来る変数はお互いを鎖のようにつないでおいて，その増減分（例えばEQUIVALENCE (A, B(7)) のように宣言された時の7に当たる値）を別に用意する table に記憶する。

EQUIVALENCE はその中に現われる変数を鎖のようにつなぐことになるが，一つの EQUIVALENCE 文によって出来る鎖が，他の EQUIVALENCE 文によって出来る鎖と交わるとき，例えば

EQUIVALENCE (A, B, C)

EQUIVALENCE (B, D, E)

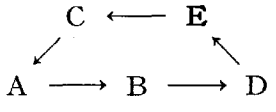
のようになるときは，鎖を B の所ではずして，そこに D, E の鎖をつないで，新しく，宣言文

EQUIVALENCE (A, B, D, E, C)

によって出来るような鎖を作る。

つまり

のような鎖を B の所ではずして



のような鎖に作りかえるのである。

この手続きの最中に，2番目の鎖が2個所以上で1番目の鎖と交わっている時には文法エラーとする。

以上の手続きを例で説明しよう。

例　COMMON A, C, D
　　EQUIVALENCE (A, B(7))
　　DIMENSION A(10, 40), B(100)

COMMON, EQUIVALENCE, DIMENSION の扱い方

name 表											
address	名前	l	r	q	c	D	Ad	L	LM	LMN	step
n_1	A				1	16					I
n_2	C				1	16					COMMON
n_3	D				1	16					をよみ終った時
n_1	A		1	1	1	16	n_4				II
n_2	C				1	16					EQUIVALENCE
n_3	D				1	16					をよみ終った時
n_4	B		1	1	1	16	n_1				
n_1	A		1	1	1	16	n_4	10	400		III
n_2	C				1	16					DIMENSION
n_3	D				1	16					をよみ終った時
n_4	B		1	1	1	16	n_1	100			
n_1	A		1	1	1	16	$l-10-1$	10	400		IV
n_2	C				1	16	$l+400$				COMMON
n_3	D				1	16	$l+401$				EQUIVALENCE
n_4	B		1	1	1	16	$(l+1-7)-1$	100			DIMENSION
											処理

l, r, q, c はいずれも name 表の 1 bit からなる情報である。

注　上の表中空白は0である。

Step I. COMMON A, C, D と読み name 表と COMMON 表に情報を書きこむ。例では偶々 COMMON 宣言が先に来たので COMMON 表の並びと name 表の並び方が一致しているが，常にはこういう形にはならない。name 表の D part にはまだ A, C, D が何次元の変数あるいは配列名となるか判らないので，最長の表の長さ16を書き込む。

Step II. EQUIVALENCE 宣言を読み，A と B とを name 表の ad part を利用して鎖でつなぐ。A は COMMON 宣言がなされたことが name 表の c bit が1なることより判るから，これと結合された B も COMMON エリアに入ることとなる。A, B に割付けられる address が A(1) と B(7) とが等しいという意味で，EQUIVALENCE 表の location 加減分にそれぞれ +1, +7 を書き込む。

COMMON 表		EQUIVALENCE 表	
table address	割付 address	table address	location 加減分
n_1			
n_2			
n_3			
n_1		n_1	+1
n_2		n_4	+7
n_3			
n_4			
n_1		n_1	+1
n_2		n_4	+7
n_3			
n_4			
n_1	l	n_1	+1
n_2	$l+400$	n_4	+7
n_3	$l+401$		
n_4	$l+1-7$		

Step III. DIMENSION 宣言を読みとり，その配列の大きさをそれぞれ L, LM, LMN part に書きこむ。

Step IV. すべての宣言文が現われた所で，まず COMMON 表をみて，COMMON 表の first-in の変数または配列から COMMON エリアにメモリを割付けていく。

COMMON 表に入っている n_1 より name 表のどの変数かを知り，それに番地を割付ける。A は EQUIVALENCE 宣言がなされて

いるから，これと結合されている B にも同様に番地を割付ける。この例では結果として B に割付けられる address が COMMON エリアの先頭 address l よりも小になるから，エラーとして検出される。このようにして COMMON をはじめに片付けてから，後はまだ処理の終っていない変数または配列を name 表からさがしだして順に変数エリアに割付ければよい。

かくして，宣言文に現われた変数または配列はすべて番地の割付けが完了する。

なお，上述の方法によれば宣言文の現われる順序には制約を置かなくてもよい（JIS 3000 FORTRAN では制限事項がある）ことを重ねて注意しておこう。

＜メモリの割付＞

単純変数と定数のメモリ割付についてのべる。subprogram 関係についてもメモリの割付を考えねばならぬが，それは **4. 3.** において考える。

定数はそれが整数型か実数型かを判定して，次々に定数 table に登録する。重複は避け，数値は絶対値だけを登録する。

変数は一回一回 name table を参照して，未登録のときにのみ登録を行なっていき，同時に，変数エリアに割付けを行なう。

変数が関数の引数として宣言されたものであるときには，引数であることを subprogram 表の Sp part の 1 bit を使って表示しておく。

引数に割付けられる working storage は，内部メモリの address を表現するに足る大きさのメモリで，2 バイトである。（**4. 3.** 参照）

statement function で用いられた引数は，その statement function を定義する statement の処理を終った時に，後で使えないようにしてしまう。これには名前を変化させてもよいし，その変化に対応する bit をつけてもよかろう。statement function の引数は，型さえ一致しているならば，プログラムのどの名前を使ってもよいという規則があるため，この処置が必要な

のである。

4. 3. Subprogram の取扱い

PHASE II における subprogram の取扱いについて述べる。

＜Subprogram が定義された場合＞

PHASE II では subprogram が定義された時には，変数エリアに関数値を蓄える working storage を一変数分 reserve する。処理プログラムを簡単にするため，FUNCTION statement と SUBROUTINE statement の取り扱いをまったく区別しないことにする。

このように割付けた address は subprogram 表に subprogram name がすでに登録されている場合にはその entry の ad_3 part に格納し，まだ登録されていない場合には name とともに新規に登録する。また，引数をいったん退避させるための working storage も割付けを行なう。

＜Subprogram が使用された場合＞

この時は **4. 4.** 以下にのべるようなシラブルを作り出すのであるが，同時に使われた subprogram がまだ subprogram table に登録されていない場合には登録を行なう。

＜statement function が定義される場合＞

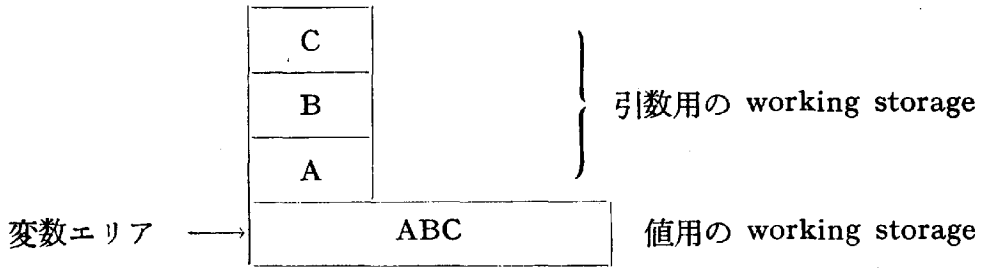
statement function は PHASE I では単に数式として分類されてくるだけなので，PHASE II では，

FUNC(A)＝A**2

のように等号の来る前に左カッコ '(' が来たときに，name 表を調べて FUNC という名前が配列名でない時に statement function であると判断する。

例　FUNCTION ABC (A, B, C)

と定義したとき



のようにメモリを割付ける。

4. 4. シラブル分解

FORTRAN statement で，その処理が PHASE III に渡されるものは以下にのべるようなシラブルに分解される。

各シラブルは次の構造を持つ。

w： text 部分の長さ（1バイト）

k： シラブルの種類（1バイト）

text： 具体的な情報（w で指定しただけのバイト数）

k の 2^0 bit　key word のとき 1

2^1　variable のとき 1

2^2　constant のとき 1

2^3　special character のとき 1

2^4　割付けられない定数のとき 1

2^5　FORMAT のとき 1

2^6　subprogram のとき 1

2^7　整数型のとき 1　実数型のとき 0

text 部分

$k=2^0$ のとき　1バイトの分類番号

$k=2^1$ のとき

D	ad	L	LM	LMN

9 バイト

D の 2^7 bit=0 ならば D には変数の次元が set される。

D の 2^7 bit=1 ならばこれは引数のシラブルを表わす。

ad, L, LM, LMN には name table の対応する部分の情報が set される。

$k=2^2$ のとき　$k=2^1$ に同じ。

$k=2^3$ のとき　1 バイトの special character それ自身 (e.g. *|/|,| etc.)

$k=2^4$ のとき

x

3 バイト

MSB の x（1 ビット）の内容が 1 ならば下（シモ）2 バイトに 2 進数表現の定数が set される。

x=0 ならば下（シモ）2.5 バイト（5 桁）に 2 進化 10 進数の定数が set される。（ステートメントナンバー等はこのケースになる。）

$k=2^5$ のとき text 部分はすべて FORMAT statement の中味である。

$k=2^6$ のとき

loc

2 バイト

loc：　Subprogram 表の位置

なおこのシラブル分解の際には statement の終了を示す special character のシラブルを conventional に作り出すものとする。

また，PHASE III でこのようにして出来たシラブルを順に読んでいって compile を行なう訳であるが，error check の方法が悪いとプログラムの終りが来ても正常に止らないで暴走する危険がある。これをなくするために，一つのプログラムの終りにはプログラムの終りを示す一つの special

character を必ずつけておいて，compile の際にこのシラブルが出て来たときにはエラーがあったものとみなすことにする。

簡単な例をシラブルに分解してみる。

例 1. STF(A)＝A

1. 01 01 14　　key word 数式の分類番号 14
2. 02 40 loc　　statement function が来たことを示す
3. 03 02 00 ad　　STF を変数とみたシラブル
4. 01 08 ＝　　＝のシラブル
5. 03 02 00 ad　　A のシラブル
6. 01 01 (end)　　1 statement の終了を示す

例 2. SUBROUTINE X(A, B)

1. 01 01 32　　SUBROUTINE の分類番号
2. 02 40 loc　　subprogram が来たことを示す
3. 01 08 (end)　　1 statement の終り

例 3. 100 FORMAT(1H0, 2I5)

1. 01 01 25　　FORMAT の分類番号
2. 03 10 00 01 00　　statement number 100
3. 08 20 1 H 0, 2 I 5)　　text 部分
4. 01 08 (end)　　1 statement の終り

5. PHASE III

5. 1. 概 要

PHASE III は PHASE I, II によって分類し変数の割付等の終了した結果を使って，実際の object program を作り出す，いわゆる code genera-

tion の機能をもつ PHASE である。

PHASE III は大きくわけて，PHASE の初期化を行なう部分，本体，終了処理部分にわかれる。本体は大きくわけて，各処理プログラムの仲介をとるふりわけ，共通ルーチンとコントロール statement 処理，入出力命令処理，数式処理の各プログラムがある。

5. では終了部処理（END statement が来たときの処理）は **5. 3.** のふりわけルーチンの説明の所でのべている。

5. 2. 初期化部分

この部分は 1 segment をなし，補助記憶装置から内部メモリに読みこまれると必要な counter, indicator 類に初期値が与えられるようにしておく。PHASE III で必要とする共通ルーチン等も，この segment に組みこんでおいて，同時に内部メモリに読みこまれる。

PHASE II の処理が行なわれた結果，どの組込み関数が使用されたかが判明するから，subprogram table をみて，この段階で library 関数の割付を行なう。組込み関数のうち SIN と COS とは普通入口は若干異なるが，本体はほとんど同じものを使用するから，一方が使用されたことは，もう一方の関数も object program に組込まれることを意味する。従って COS が使われたときには常に SIN も使われたような扱いにすることが必要である。組込み関数を object program 内に組込むのはかなり面倒な考慮が必要だが machine dependent な面が多いので説明を省略する。

ここで clear する indicator 類は AIND, ASTS で，初期化する counter 類は object program の割付 counter, ステートメントナンバ表の counter, DO table の counter である。

次のふりわけルーチンにコントロールを渡す前にプログラムの最初のシラブルをためしよみして，それが subprogram の宣言のシラブルならば sub-

program appeared indicator を ON に set し，その情報をいったん save してふりわけに行く。

5. 3. ふりわけ，共通ルーチン，END 処理

内部メモリが小さい場合には，全部の statement の処理プログラムを一時に内部メモリにいれておくことはできないから，必然的にそれらの処理プログラム間の仲介をするプログラム——ふりわけルーチンが必要となる。はじめにふりわけルーチンと END 処理について述べる。

このルーチンは最初に 5. 2. の PHASE III の初期化ルーチンからコントロールを渡され，PHASE II の出力結果であるシラブルの分類番号を次々によみ，各処理プログラムにコントロールを渡す。

他にふりわけルーチンは，メモリ割付 counter の set または reset，また compile 時に使用する各種 table の overlay control，statement function の前後処理，未定義 subprogram の芋ずる定義，若干の statement の compile 等を行なう。

以下このルーチンの大体の流れを描写してみよう。ふりわけルーチンには 2 つの entry point がある。その一つは statement の最初から compile する場合の入口であり，それを ENTRY 1 と呼ぶ。もう一つは statement の途中から compile する場合の入口であり，これを ENTRY 2 と呼ぼう。

以下 ENTRY 1 に入った時にはコントロールは (1) に行き ENTRY 2 に入った時には (2) に行くものとする。

(1) ENTRY 1. 直前に処理した statement が statement function ならば (8) に行く。

直前に処理した statement が DO の端末文で，しかも DO table に処理すべき DO 文の情報が残っているときにはステートメントナンバ表を内部

メモリによみこんで DO の end 処理プログラムにコントロールを渡す。

statement の分類番号をよみ，いったん記憶する。

ステートメントナンバが1～5カラムにある場合には，ステートメントナンバ表を内部メモリによみこみ，ステートメントナンバ定義ルーチンにコントロールを渡す。

(2) ENTRY 2. いま処理しようとしている statement が数式の場合には(7)に行く。

END statement の場合には END 処理部(6)に行く。

(2)′ いま処理している statement がプログラムの初めのものならば，プログラムの初期部(4)に行く。

(3) ＜ふりわけ部＞ 分類番号に従って処理プログラムにコントロールを渡す。その際，READ 命令，WRITE 命令のようにステートメントナンバの処理を伴なう statement の処理プログラムにコントロールを渡す前にはあらかじめステートメントナンバ表を内部メモリによみこんだ状態にしておく。

(4) ＜プログラム初期化部＞ object program の割付 counter AWIX を reset し，プログラムが常駐か否かに従って，address 補正 counter ADIF の内容を適当に set する。(object program は counter AWIX の示す位置以下に作るが，その実際の実行 address location は，この値に ADIF の内容を加えたものである。)

現在 compile 中のプログラムが subprogram ならば，**5. 2.** の初期化部分で待避した subprogram の情報を recover する。subprogram 表を内部メモリによみこむ。

この subprogram が他の program ですでに未定義のまま使用されたことがある場合には，subprogram の芋ずる定義部(5)へ行く。その必要がないときには subprogram 表を定義してふりわけ部(3)へ行く。

現在 compile 中のプログラムが subprogram でなければ，main program が初めて現われたことになるから，これから出力する object program の page 数を記憶して，ふりわけ部 (3) に行く。

(5) ＜subprogram の芋ずる定義部＞ subprogram 表の中の芋ずる情報をいったん待避し，subprogram 表を完全に定義する。芋ずる情報を逆に順々にたぐっていき今迄使用されたこの subprogram に関する情報を正しく書きこむ。ふりわけ部 (3) へ行く。

(6) ＜END 処理部＞ いま compile し終ったプログラムが常駐 subprogram であった場合には，常駐エリアの counter AMIX の内容を (AWIX)+(ADIF) に等しくする。

現在途中まで compile した結果の入っている object を 1 page として，完結させて出力する。

未定義のステートメントナンバを印字し，指示があれば name 表，ステートメントナンバ表を印字する。

次のプログラムの compile のために再び PHASE I に戻る。

(7) ＜statement function 検出部＞ 次のシラブルをためし読みし（もう一度シラブルをよんだ時に同じシラブルをよんでくるようなよみ方）それが subprogram のものであれば statement function が現われたことになるから，statement が現われたということを indicator に set して，subprogram 表を定義し，ふりわけ部 (3) に行く。statemant function 以外の数式の場合には (2)′ へ行く。

(8) ＜statement function 終了部＞ statement function の main ルーチンへの復帰命令を出力し，途中迄出来上っている object program を完結

(注) subprogram に link するには subprogram の引数に割付けられた address, subprogram のページ数，subprogram の入口 address，引数の個数，引数そのものの情報が必要である。このうち，subprogram を未定義で使用したときどうしても判らない情報は，初めの 3 つであるから，この subprogram の芋づる定義においては，この 3 つの情報（6 バイト）を subprogram 表に定義した情報を基に書きこむのである。

した 1 page として出力する。

statement function が現われたことを示す indicator を OFF にして ENTRY 2 へ行く。

以上はふりわけまたは分類という面からみたふりわけルーチンの大体の流れであるが，他の簡単な処理もこのルーチンに同時に負担させてしまうのが良い。

RETURN 命令は上の statement function の終了部の処理とほぼ同様なので，共通のルーチンが使える。

object program は 3 つのモードのいずれかの状態で実行されるが，このモードを変化させる可能性のある statement の compile を行なう時にはその各 statement の前に，execution time にモードを判定して，必要に応じてモードを変化させる object program をつけなければならないが，これもふりわけ部で行なうことにする。

以上の説明では明確に述べなかったが，以上のふりわけの仕事や，それからもっと一般に code generation に関して必要とされる共通ルーチンがあるので，それらについて少し述べよう。

［ページ切換ルーチン］ このルーチンは一つのプログラムが，compile の最中に object program が 1 page より大きくなったときに自動的に page を切換えるための SR である。

code generation の際には，あらかじめ何バイトの命令を出力するかということをこの SR に教えてやることにする。この SR は object program の作成エリアの counter AWIX の内容と，作成エリアの上限とを比較して，与えられたバイト数だけの object program が，1 page のエリアに入り切

らないときには，その段階迄に作られた object program を完結した 1 page のプログラムとして出力し，object 作成エリアの counter AWIX, address 補正 counter ADIF の内容を reset し，pageの counter を 1 raise する。

［シラブル read SR］ PHASE II で作られたシラブルを 1 個ずつ取り出すための SR。ためしよみもしくは空よみの機能を持ち，さらによみ過ぎが生じたときにはエラー処理を行なうものとする。(4. 4. 参照)

筆者は COBOL compiler [4] の作成において，これと同様の SR を作成し，利用したが，そこでは空読みの機能の代りに，くり返しよみともいえる機能をもたせた。つまり，一度読んだものを再びよむことができるようにしたが，これよりも空読みの機能の方が，種々の点で compile には都合がよいように思われる。

［code 出力 SR］ 与えられた情報に基づいて，code または情報を object program 作成エリアに植えつけていく SR。

［object program access SR］ subprogram の手ずる処理の際に生じたように，時にはすでに 1 page として出力した object program の中味を手ずる処理をするために変更しなくてはならない場合があるが，いったん 1 page としてまとめて出力した object program を再び内部メモリによみこんだり，またはその逆を行なうための SR。

参 考 文 献

［1］ 日本電子工業振興協会：日本工業規格原案 電子計算機プログラム用言語 FORTRAN 3000 41年10月。

［2］ 日立製作所 HITAC 5020, 5020 E/F FORTRAN (HARP) 5020-3-005-2。

［3］ 森口繁一 FORTRAN IV 入門 東京大学出版会 1915年。

［4］ 穂鷹良介 Automatic Coding について III ——COBOL Compiler (1)及び (2)—— 商学討究第18巻1, 2号。