ALGŌL Compiler の開発 (1)

稲田信幸

ALGŌL は FŌRTRAN と同様に科学技術計算向きのプログラミング言語である。これらの言語は機械語やアセンブラ言語に対してコンパイラ言語と呼ばれる。ALGŌL は FŌRTRAN 程使用頻度が高くはないが現在では世界共通の言語として国際標準規格が制定され，国内でも JIS 規格が制定されている。

一般にプログラミング言語で書いたプログラムのことをソースプログラム (SP) といい，翻訳されて機械語となったものをオブジェクトプログラム (ŌP) という。ここで SP から ŌP への変換は人間がする訳ではなくコンパイラと呼ばれるプログラムが担当する。従って一般にプログラマは機械語やコンパイラのことを知らなくても，問題に一番適したプログラミング言語を使用して計算手続きを記述すればよい。

本稿は小樽商科大学の ŌKITAC-5090H 計算機システム（以下 H と略す）用に開発した ALGŌL コンパイラの内部仕様書である。

H には現在に至るまでオペレーティングシステムは存在していない。それゆえ FŌRTRAN コンパイラも CŌBŌL コンパイラも自分だけのモニタを有して，自分の言語のプログラムに対してバッチ処理を可能にしている。

この ALGŌL コンパイラも同様なモニタのもとで動作するようになっている。厳密には一致していないけれど文法としては JIS 規格の 3,000 程度のものを選んだ。開発には 2 名[(1)]で約 2 カ年を費した。この ALGŌL コンパイラを作るのに使用したアセンブラ言語は筆者がそのために開発した CARD-IPH（従来の紙テープベースの IPH を一部訂正し，機能的にかなり拡張したものである）で，これを用いて作成したコンパイラはカード約 5,000 枚を要している。コンパイラはモニタ，PASS-I, PASS-II, 実行ルーチンから構成されているので，その順に沿って記述する。

---

(1) 富岡秀広（現在，北海道銀行札幌支店勤務）と筆者の 2 名が戸島・清水川ゼミナールで開発した.

目 次

1. モ ニ タ

1.1. 概　　要

ここでのモニタはオペレーティングシステム (ŌS) でいうモニタとは厳密に区別して考えなければならない。以下モニタとは筆者の作成した ALGŌL コンパイラのジョブ管理，フェイズ管理及び入出力管理をするプログラム群のことである。ジョブ管理及びフェィズ管理はモニタカードによって行なわれる。入出力管理は簡単なもので，ŌS でいうデータ管理ではなくて単に入出力を統一して，入出力 ōperation の end 待ちを極力少なくするためのものである。モニタにジョブ管理を行なわせることによってジョブの連続処理が可能になる。PASS-I 及び PASS-II の処理フェイズの各処理プログラムや実行フェイズのオブジェクトプログラムは主記憶に一時に格納できないので MT に入れておき，フェイズ管理に MT の handling を行なわせて主記憶をオーバレイして使用する。適当な情報をモニタカードでモニタに与えるとソースプログラムを翻訳するだけでなく，ソースプログラムのリスティングやオブジェクトプログラムのリスティングも行なう。

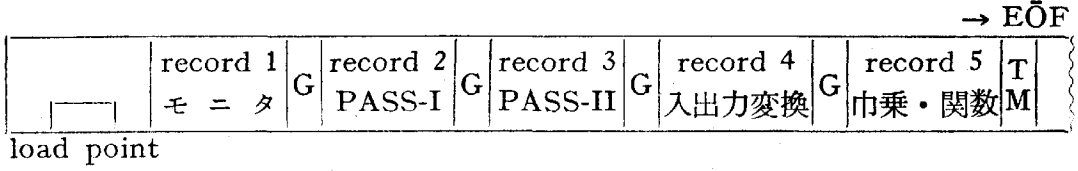
モニタはシステムテープ(2)の最初の record で，システム・イニシャルローダ(3)によって主記憶に読み込まれ，各フェイズの実行後も破壊されることはない。

1.2. ジョブ管理・フェイズ管理

ジョブ管理とフェイズ管理に言及するに当っては，モニタカードの種類とプログラムカードデックの作り方を述べなくてはならない。

---

(2) システムテープは次のような構成になっている.



(3) システムテープの最初の record を主記憶へ読み込み，それに control を渡すように作られている．ハードのブートストラップを使用して PTR から読んでいる.

モニタカードは5種類あり，その意味は表1のとおりであり，更にモニタカード1のnによって表2の動作をする。

表1　モニタカードの種類と意味

1. ／ MŌNITŌR ALGŌL-IT (n)　　モニタ動作開始とリスト指定.
2. ／ 何が Punch してあってもよい.　ID カード.
3. ／ END ŌF SŌURCE　　ソースプログラムの終り.
4. ／ EŌF　1 JŌB ファイルの終り.
5. ／ SYSTEM END　　バッチ処理の終了.

表2　リ ス ト 指 定 表

n	オブジェクトの実行	ソースのリスト	シンボルテーブルのリスト	オブジェクトのリスト	中間語のリスト
1	○	○	○	○	○
2	○	○	○	○	×
3	○	○	×	×	×
4	○	×	×	×	×

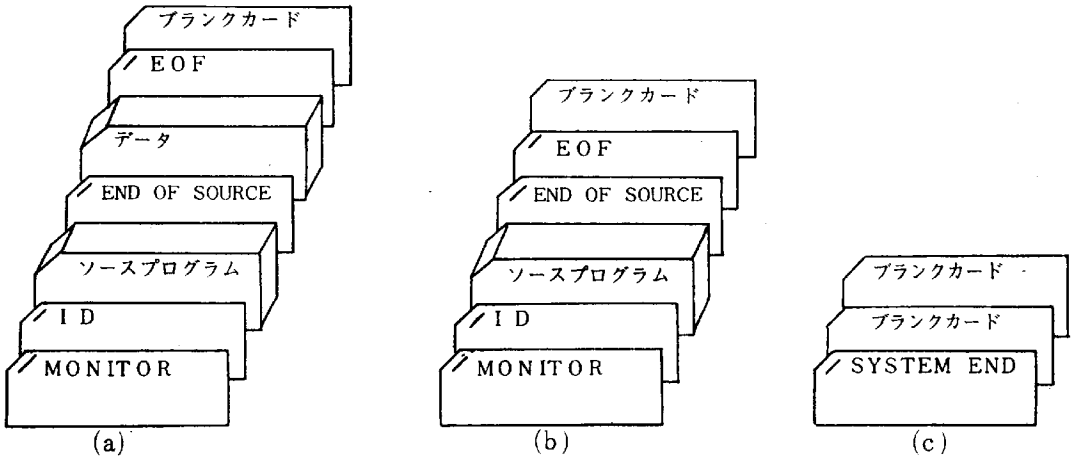


図1　プログラム カード デック

カードデックはデータのある図1(a)とデータのない図1(b)を複数個まとめて，最後のブランクカードのかわりに図1(c)のカード群をつけて構成される。

フェイズは3つある。フェイズ1は PASS-I の実行，フェイズ2は

PASS-II の実行，フェイズ3はオブジェクトプログラムの実行をそれぞれ行なう。フェイズ毎にオーバレイしているので，フェイズがかわると主記憶の割当ても図2のようにかわる。

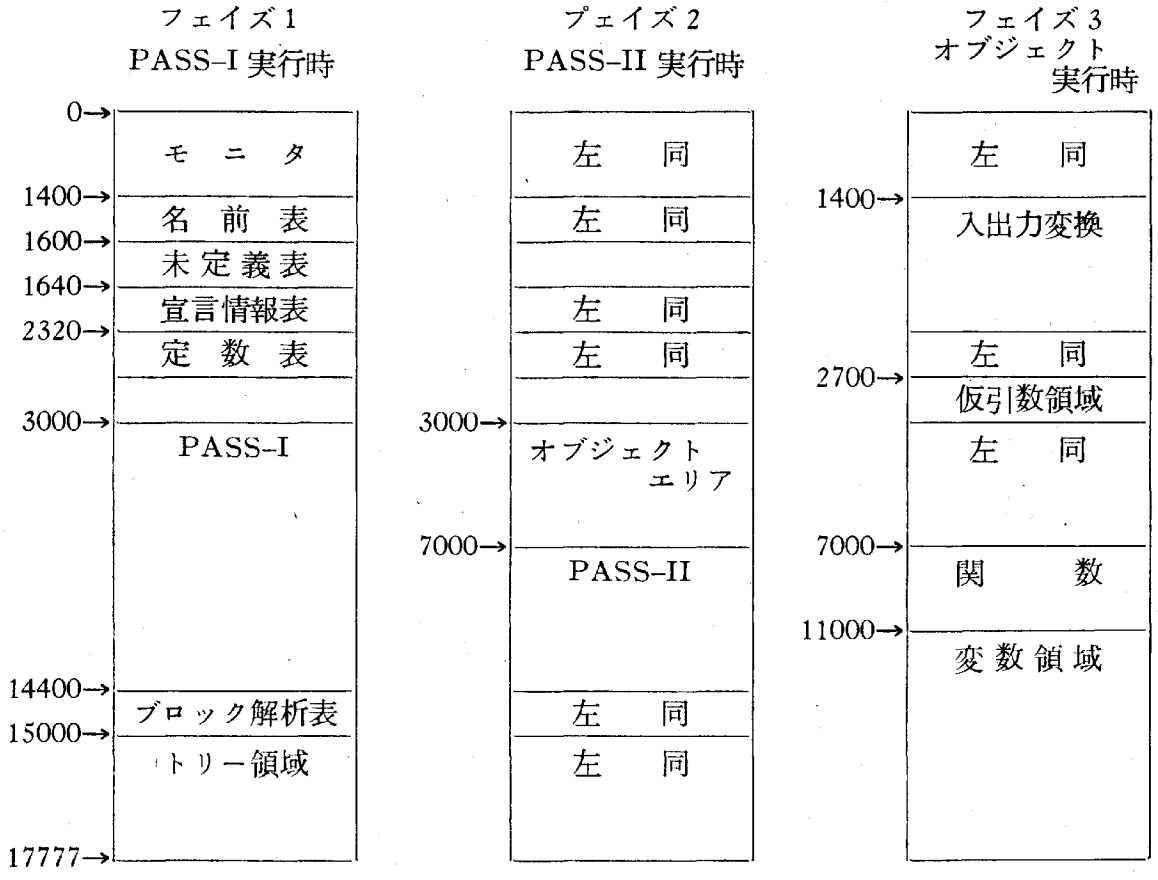


図2 各フェイズにおける主記憶割付（オクタル表示）

H に備わっている10個のプログラムスイッチ（PS）のうちいくつかは表3の意味に使用している。

表3 プログラムスイッチ割当て

PS (1) オブジェクトプログラム実行中 on.
PS (6) モニタカードエラーの時 on.
PS (8) 実行時に ／ EOF カードが現われると on.
PS (9) on の時ソースプログラムのリストをとる.
PS (10) 入力サブルーチンがモニタカードを読んだ時 on.

ジョブ管理・フェイズ管理の動作の概略は次のとおりである。

stage 1. モニタの初期化。

stage 2. ジョブの初期化。

stage 3. モニタカード1及び2の判定。

stage 4. PASS-I を主記憶に読み込んで，それに control を渡す。

stage 5. モニタカード3の判定。

stage 6. PASS-II を主記憶にオーバレイして読み込み，それに control を渡す。

stage 7. 実行ルーチンを主記憶にオーバレイして読み込み，PS(1) を on にしてからオブジェクトプログラムに control を渡す。

stage 8. モニタカード4の判定。

stage 9. モニタカード5の判定をし，モニタカード5の時は最後の **stage 10.** へ行き，ちがう時は **stage 2.** へ行き，これを繰り返えす。

stage 10. システムテープを巻き戻し，ジョブ及びモニタの後処理をする。

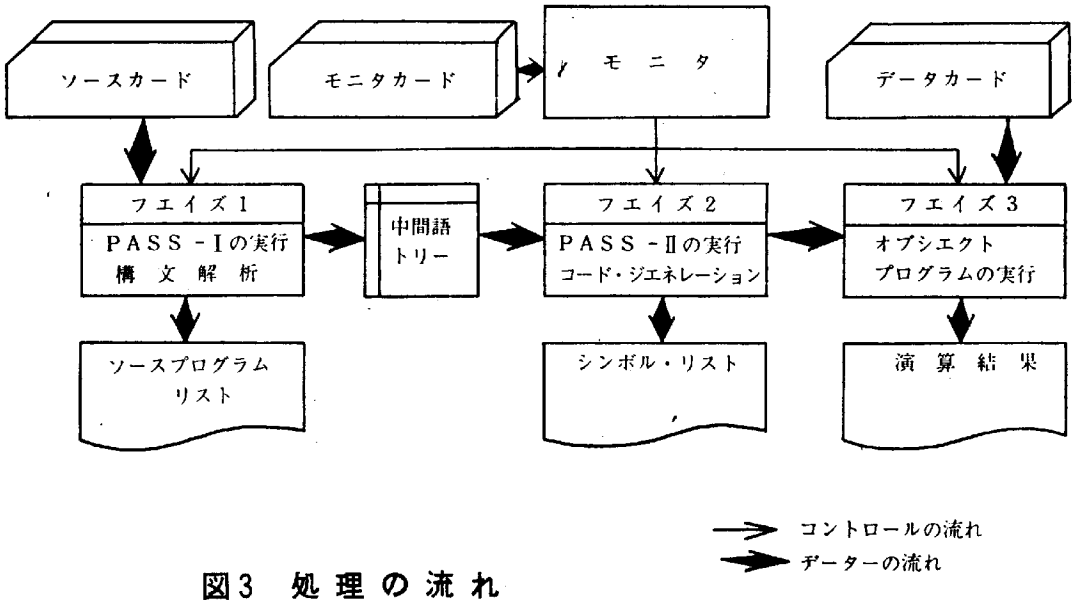


図3 処理の流れ

1.3. 入出力管理

本 ALGŌL コンパイラでは入力ファイルはカードリーダに対応し，1 record はカード1枚（80字）に相当する。出力ファイルはラインプリンタで，1 recōrd は1行（120字）に成当する。入力ファイルからのデータの転

送および出力ファイルへのデータの転送は入出力管理のサブルーチンを介して行なわれる。すなわちモニタがモニタカードを読む時，PASS-I がソースプログラムを読む時，オブジェクトプログラムが実行されてデータを読む時のいずれの場合でも入力サブルーチンを介して行なわれる。出力の場合も同様に出力サブルーチンを介して行なわれる。こうすることにより入出力管理の方で buffering することが可能となり，入出力命令の operation end 待ちを少なくしうる。

つぎに，入出力管理を構成する7つのサブルーチンの説明をする。

(1) open-in-file

システムが稼動を始める時に call される。先読み buffer にカード1枚を読み込む。

(2) close-in-file

システムが終了する時に call される。カードリーダの operation end 待ちを行なうだけである。

(3) in-line-feed

システムの稼動中，入力ファイルの access は常にこのサブルーチンを介して行なわれる。主な動作は先読み buffer の80字を data buffer に転送して先読み buffer にカードを読み込む。data buffer の内容がモニタカードの時はプログラムスイッチ (PS)10 を on にしてモニタをはじめとして各処理プログラムに知らせる。

(4) open-out-file

ジョブの初期化の時に call される。出力 buffer の内容を空白にし，ライン及びカラムカウンタを0にする。

(5) close-out-file

ジョブの後処理の時に call される。出力 buffer に出力すべき文字が入っていれば出力し，用紙をホームポジションまで skip する。

(6) out-line-feed

システムの稼動中，出力 buffer の内容を出力ファイルへ書き出す時は常

にこのサブルーチンを介して行なわれる。出力 buffer の 120 字を data buffer に転送し，ラインプリンタに印字し1行改行する。そして出力 buffer には空白を入れて，ラインカウンタを 1 つ上げ，カラムカウンタを 0 にする。

(7) outstring

文字 string を印字したい時に call される。outstring はモニタをはじめとして各処理プログラムで随時使用されるので常駐エリアに置かれている。引数の記号列を出力 buffer に詰めていく。

2. PASS-I

2.1. 概　　要

PASS-I の主な仕事はソースプログラムを読み，構文解析をしてその結果を中間語のトリーに変換して PASS-II への橋渡しをすることである。PASS-I 自体モニタから呼ばれる大きなサブルーチンでたくさんの小ルーチンから成っており，中には自分自身を呼ぶことのできる回帰的なサブルーチンもある。ソースプログラムを読んだり，ラインプリンタにリストやエラーメッセージを印字したりする仕事は入出力管理の入出力サブルーチンで行なわれることは **1.3.** で述べたとおりである。

本 ALGŌL コンパイラはソースプログラムを読み込んだ時点でリストをとり，シラブルに分解して，そのシラブルを使用して構文解析を行なっている。構文解析をしている途中で名前がどのブロックで使用され，それがどういう意味で使用されているか調べるために，ブロック解析表と宣言情報表を用意してある。

その他定数表，名前表，未定義法等の表が PASS-I だけでなく PASS-II にも共通に使用されている。

ALGŌL の文法は BNF で完全に表現しうる。しかし計算機の外部コードに対応させることのできない基本記号を ALGŌL はもっているので計算機で使用するためにハードウェア表現を用いることにする。

2.2. 表とそのデータ構造

2.2.1. 名 前 表

ソースプログラムに現われた名前は同一パターンのものが一度だけここに登録される。1 word 42 bit の中には 7 文字入れることができる。本 ALGŌL コンパイラでは 7 文字に満たない時は右に詰めて左に 0 を入れている。

C_1	C_2	C_3	C_4	C_5	C_6	C_7

C_n : 各 6 bit.

2.2.2. 未 定 義 表

行先の決まらない **go to** 文が現われる毎に作られる。未定義で使用される名札はブロック後処理の結果，あるいは名札が宣言されることによって行先が確定するので未定義表からはずされる。

PASS-I の終了した時点で，いまだに行先の決まらない **go to** 文がある時はエラーとなる。

Z	S	T	X	L

Z: 1 bit. 未使用で常に 0。

S: 6 bit. BLC（ブロックカウンタ）の値が入る。

T: 7 bit. BGC（ビギンカウンタ）の値が入る。

X: 14 bit. 名前が名前表に登録されている番地が入る。

L: 14 bit. 未定義 **go to** トリーの節の作られた番地が入る。

2.2.3. 定 数 表

ソースプログラムに現われた数と記号列が登録され，定数表は実行時にもそのまま使用される。数については絶対値が一度だけ登録され，整数，実数のいずれにも 1 word ずつ割当てられる。記号列は 1 word に 7 文字入り，連続した番地に格納されている。

2.2.4. ブロック解析表

ブロックの先頭で宣言される毎，あるいは名札が〈名前〉：の構文でもって宣言される毎に **2.2.5.** の宣言情報表と共に作られる。このブロック解析表はそのブロックが終わると clear してもよいのだが，本 ALGŌL コンパイラでは 1 bit のフラグ（Z パート）を設けて clear しないことにして，あとでブロック解析表と宣言情報表をリストできるようにしてある。しかしその反面，解析表の大きさしか宣言できないことになった。

Z	S	T	X	Y

Z： 1 bit. ブロック後処理フラグ。

S： 6 bit. BLC（ブロックカウンタ）の値が入る。

T： 7 bit. BGC（ビギンカウンタ）の値が入る。

X：14 bit. 名前が名前表に登録されている番地が入る。

Y：14 bit. 宣言情報表の番地が入る。

2.2.5. 宣言情報表

この宣言情報表が **3.2.4.** のブロック解析表と対になっていることはすでに述べた。名札の宣言，型の宣言，手続きの宣言によって名前に対する種々の情報がもり込まれる表である。宣言によってすべての情報が埋められる訳ではなく，PASS–II の時に決まるものもある。(4) 本 ALGŌL コンパイラでは配列の計算を簡単にし，次元数を 2 次元に限定したため **3.2.4.** のブロック解析表と対にすることができた。添字が実行時に定義域内にあるかどうかを check するためには上下限の値を実行時まで覚えていなくてはならない。

D	K	M	A	B	C	ADD

D： 7 bit. 未使用。

(4) 例えば名札の番地や手続きの入口の番地は実際にオブジェクトを作っている時に決まる.

K: 1 bit. A=3 & K=1 関数。

A=2 ... K=1 2次元, K=0 1次元。

M: 14 bit. A=3 パラメータの個数。

A=2 定数 (d−c+1) の値の入っている番地が入る。

A: 2 bit. =0 単純変数。

=1 名札。

=2 配列。

=3 手続き。

B: 2 bit. =0 空。

=1 string.

=2 実数。

=3 整数。

C: 2 bit. =0 引数でない時。

=2 引数で名前替えの時。

=3 引数で値とりの時。

ADD: 14 bit. A=0 割付番地。

A=1 名札の立てられた命令の番地。

A=2 基準番地。

A=3 入口の番地。

2.3. シラブル分解

2.1. でも述べたように ALGŌL 言語を使用してプログラムを記述する時にはハードウェア表現を用いることになるので, 当然シラブル分解するプログラムはハードウェア表現法に基づいて書かなければならない。本 ALGŌL コンパイラでは表4のハードウェア表現に基づいてシラブル分解している。従ってハードウェア表現を変えるとシラブル分解法も変えなくてはならない。シラブルは区切り記号, 名前, 数, 記号列に分けて分類する。シラブルを使って構文解析をするので, そのデータ構造は統一されていた方が便利で

ある。

1シラブルは 1 word で次のような構造をしている。

	E		F

E: 9 bit. シラブルの区別で大分類。

=0 区切り記号。

=1 名前。

=2 符号のない整数。

=3 符号のない実数。

=4 記号列。

=5 予約語[5]。

F: 14 bit. 小わけコードやポインタが入る。

E=0 区切り記号の種類を表わす小わけコード。

E=1 名前表に登録してある番地。

E=2, 3, 4 定数表に登録してある番地。

E=5 予約語の情報表の番地。

このようにソースプログラム上のすべての基本記号は統一されたシラブルになる。シラブル分解時には1文字ずつ scan してゆかねばならない。シラブル分解をするサブルーチンの中には名前を処理するサブルーチン，数を処理するサブルーチン，記号列を処理するサブルーチン及び区切り記号群[6]を処理するサブルーチンが含まれる。

表4のハードウェア表現に沿ってシラブル分解をするには，1文字で区切り記号を表現する以外のものを調べるとよい。第1文字目に英字が来ている時は名前であるし，数字が来ていれば明らかに数となる。第1文字目が *，／，(の時は必ずしも *，／，(とは限らない。第2文字目がそれぞれ *，

(5) 本 ALGOL コンパイラでは標準関数，変換関数及び入出力のための手続きを表わす名前を予約語として，宣言できないことになっているが一番外側のブロックを仮想して宣言されたことにした方が処理が簡単になる．

(6) 区切り記号の中でもハードウェア表現で・〈文字列〉の形をしたものを指す．

)，／ の時はそれぞれ ↑，]，[の意味になる。

．（ピリオド）の時は一番複雑で，シラブル上では ． は存在しない。．はすべて数に含まれてしまう。しかし ． で始まるシラブルには，記号列，：（コロン），区切り記号群があるので，それらを区別するには第2文字目で行なえばよい。数字の時は数となり，英字の時は区切り記号群，10 の時は記号列，続けて ．の時は：とそれぞれ分解することができる。

名前を処理するには第1文字目が英字かどうか調べて，次の文字に英数字が来ていたら左に 6 bit ずつシフトしていって名前を組みたてる。7文字でいっぱいになったあとに英数字が続く時は読み捨てる。名前表の中に同じパターンのものがない時は登録し，その番地がシラブルの F パートに入れられる。表に既に登録されている時はその番地がシラブルの F パートに入れられる。

数を処理するには整数モードで数を作っていき，．と 10 以外のもので終ったら整数となる。．が整数モードの次に来ていたら今度は小数モードで数を作っていき，10 が整数モード及び小数モードの次に来たら，指数部のある実数なので前の結果を一時退避し，新たに整数モードに入って数を作り上げる。こうして作られた数は整数，実数にかかわらず 1 word に入れられている。名前の場合と同様に定数表になければ登録し，定数表に登録されている番地がシラブルの F パートに入れられる。

記号列を処理するには1文字ずつ scan していき，10 と ．が続けて現われるまで 1 word 7字詰めては定数表に格納していき，最後の文字のあとに“77”（オクタル表示）を付け加えて定数表に格納した先頭の番地をシラブルの F パートに入れる。

区切り記号群を処理するには ．〈文字列〉の構文のものを小わけコードとの対応表にしておき，等しいものを対応表から捜してきてそれに対応する小わけコードをシラブルの F パートに入れる。

表4　ハードウェア表現表

a ..... z	A ..... Z	[	(/ (4)
0 9	0 9	]	/) (5)
+	+ (19)	‘	· 10
−	− (20)	’	10 ·
×	* (21)	**go to**	.GŌTŌ (10)
/	/ (22)	**if**	.IF (15)
↑	** (23)	**then**	.THEN (16)
<	.LT (24)	**else**	.ELSE (17)
≦	.LE (25)	**for**	.FŌR (11)
≧	.GE (26)	**do**	.DŌ (12)
>	.GT (27)	**step**	.STEP (13)
=	.EQ (28)	**until**	.UNTIL (14)
≠	.NE (29)	**begin**	.BEGIN (0)
,	, (8)	**end**	.END (1)
.	.	**integer**	.INTEGER (30)
10	10	**real**	.REAL (31)
:	.. (9)	**array**	.ARRAY (32)
;	; (7)	**procedure**	.PRŌCEDURE (6)
:=	= (18)	**string**	.STRING (33)
␣	␣ (空白)	**value**	.VALUE (34)
(	((2)		
)	) (3)		

(n)：nは区切り記号の小わけコード

2.4. 中間語を表現するトリー

トリーは節（ノード）と呼ばれるものの集まりであって，主記憶上ではひとつの大きな配列（トリー領域）として表現される。

節には節の種類を表わす部分と2つのポインタ部分から成っている。節は次のようなデータ構造になっている。

N	P	α	Q	β	R

N：　6 bit.　未使用。

P：　6 bit.　節の種類，分類コードは表5を参照のこと。

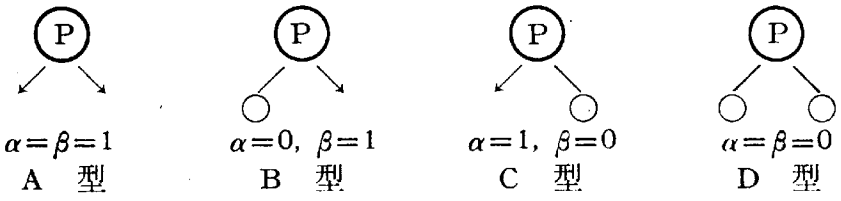
表5 節の情報分類表

コード (オクタル)	情報分類	A型	B型	C型	D型	コードに対応するシラブル	節を作る解析ルーチン他
00	**blk**	○	○	○	○	**begin**	ブロック
02	(	○	×	×	×	(	宣言，手続き宣言
03	)	×	○	×	×	)	手続き文
04	[	○	×	○	×	[	添字付変数
06	**proc**	×	○	×	×	**procedure**	宣言，手続き宣言
10	,	○	○	○	○	,	添字付変数（2次元）
11	:	○	×	×	×	:	名　札
12	**go to**	○	×	×	×	**go to**	飛越文
13	**for**	×	○	×	○	**for**	繰返文
15	**step**	×	○	×	○	**step**	〃
17	**if**	×	○	×	○	**if**	条件文
20	**then**	×	○	×	○	**then**	〃
22	:=	○	○	○	○	:=	代入文，両辺が同じ型
23	$+_I$	○	○	○	○	+	算術式，両辺整数型
24	$-_I$	○	○	○	○	−	〃
25	$\times_I$	○	○	○	○	×	〃
26	$/_I$	○	○	○	○	/	〃
27	$I\uparrow I$	○	○	○	○	↑	〃
30	<	○	○	○	○	<	比較式，条件文
31	≦	○	○	○	○	≦	〃
32	≧	○	○	○	○	≧	〃
33	>	○	○	○	○	>	〃
34	=	○	○	○	○	=	〃
35	≠	○	○	○	○	≠	〃
44	:=	○	○	×	×	なし	繰返文，制御変数代入の :=
45	,	○	○	×	×	〃	実パラメータ，記号列
46	,	○	○	×	×	〃	〃　名前替え，配列名
47	,	×	×	○	○	〃	〃　〃　，添字付変数
50	,	○	○	×	×	〃	〃　〃　，単純変数or数
51	,	○	○	○	○	〃	〃　値とり，変換なし
52	,	○	○	○	○	〃	〃　〃　，実数化有り
53	)	×	○	×	×	〃	関数呼出，手続き文と区別
54	:=	○	○	○	○	〃	代入文，整数化必要
55	:=	○	○	○	○	〃	代入文，実数化必要
56	**文 or st**	○	○	○	○	〃	ブロック
63	$+_R$	○	○	○	○	〃	算術式，両辺実数型
64	$-_R$	○	○	○	○	〃	〃　〃
65	$\times_R$	○	○	○	○	〃	〃　〃
66	$/_R$	○	○	○	○	〃	〃　〃
67	$R\uparrow I$	○	○	○	○	〃	〃　実数型↑整数型
70	$I\uparrow R$	○	○	○	○	〃	〃　整数型↑実数型
71	$R\uparrow R$	○	○	○	○	〃	〃　実数型↑実数型

Q: 14 bit. 左のポインタで，α が on の時はデータへのポインタを表わし，α が off の時は他の節へのポインタを表わしている。

R: 14 bit. 右のポインタで，β については α と同様である。

以上のようなデータ構造になっている節は P パートが同じであっても，α 及び β が on か off であるかによって次の4つの型をとることができる。それをそれぞれ A, B, C, D 型と呼ぶことにする。

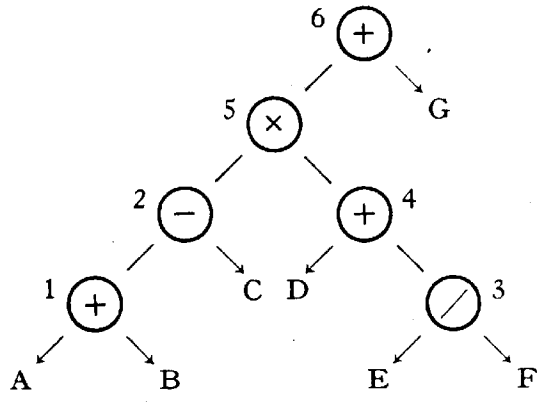


節の ポインタの中で 特殊なものとして 何も指さないものとして nil がある。この時は，α と Q 及び β と R の 15 bit がすべて on で表わしている。

中間語をトリーにしたのは PASS-II での処理が簡単になるからである。本 ALŌGL コンパイラでは実際，節の作られる順と処理される順は同じである。

（例）例として次のような算術式を考える。

(A+B−C)×(D+E/F)+G をトリーで表わすと，

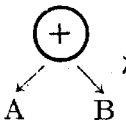


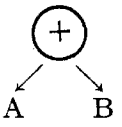
となる。

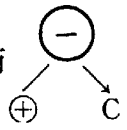
(1) 作成順は以下の通りである。

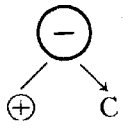
例にあげた算術式は次のようにシラブル分解される。

始 | (| A | + | B | − | C |) | × | (| D | + | E | / | F |) | + | G | 終

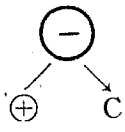
始, (, A, +, B は順に読まれてそのままスタックされる。次に−を読んだ時 A＋B の節  ができる。この時のスタックの内容は次の通りである。

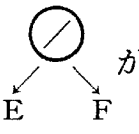
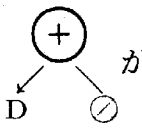
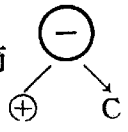
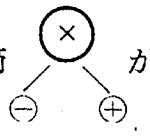
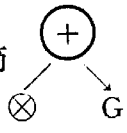
始, (,  , −

この状態から更に C,) を読むと，今度は節  が作られて，(と) は抹消される。この時のスタックの内容は次の通りでてる。

始, 

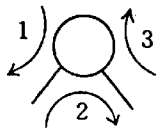
更にシラブルの ×, (, D, +, E, /, F は順にスタックされる。この時のスタックの内容は次の通りである。

始,  , ×, (, D, +, E, /, F

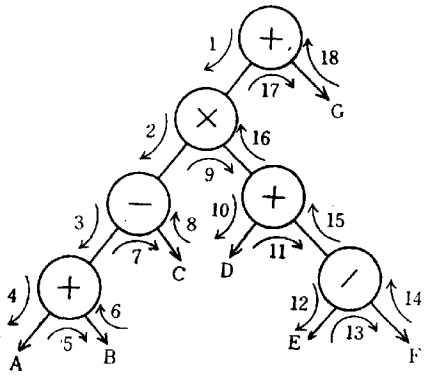
次に) を読むと，節  が作られ，更にこの節とスタックされている D と + でもって節  が作られ，この節と二番目に作られた節  と ×でもって節  が作られる。そしてスタックの内容は 始 とこの節だけになり，+ と G が読まれてスタックされ，最後の 終 を読んだところで最後の節  が作られ，始 と 終 は捨てられる。すなわちトリーの各節のところの数字が作られる順を表わしている。

(2) 処理順は次の通りである。

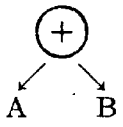
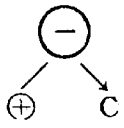
本 ALGŌL コンパイラではトリーの通り方は左回りの場合だけを考える。1つの節は必ず3つの通り方がある。



本 ALGŌL コンパイラでは1と2は単に通り抜けるだけで何も処理しないことにし，3の時にその節に対する処理をすべてすることにしている。先程の例のトリーの通り抜ける道筋を番号で表わすと，



となる。

最初1から5までは何ら処理されず，6ではじめて  に対するオブジェクトが作り出される。次は8のところで  のオブジェクトが作られ，14, 15, 16, 18 でそれぞれの節に対するオブジェクトが作られる。

この処理される順は節が作られていく順と全く同じである。

2.5. 回帰的手法について

ALGŌL の文法では回帰的に定義されている部分がかなりあるので，翻訳プログラム自体回帰的に処理しなくてはならない。すなわち recursive call を許すサブルチーンはそのサブルチーンに入った時は，以前使用されていたデータを退避させて，return 時にデータを元に戻すとよい。しかし何重にも自分自身に入る場合があるので使用されているデータをスタックすることになる。サブルーチンの入口では push down data をし，出口では pop up data をすればよい。この時，return address もデータに含めないと元に戻

れないことになる。

本 ALGŌL コンパイラでは AR（アリスメティックレジスタ）の0から6までを各回帰的サブルーチンではデータとして使用することに統一して簡略化を図った。

2.6. 構文解析

2.6.1. ブロックの解析

ブロック解析の概略は図4のようになる。ブロックは **begin** と **end** の間にある宣言と文の解析を行ない，それらの節を **blk**(7) と **文**(**st**)(8) という節でひとつにまとめる。名前の有効範囲はブロックの内側だけであるので，ブロックカウンタ（BLC）を設けてブロックの重なり具合を調べている。宣言の解析，文の解析及びブロック後処理はそれぞれの処理ルーチンでもって行なう。

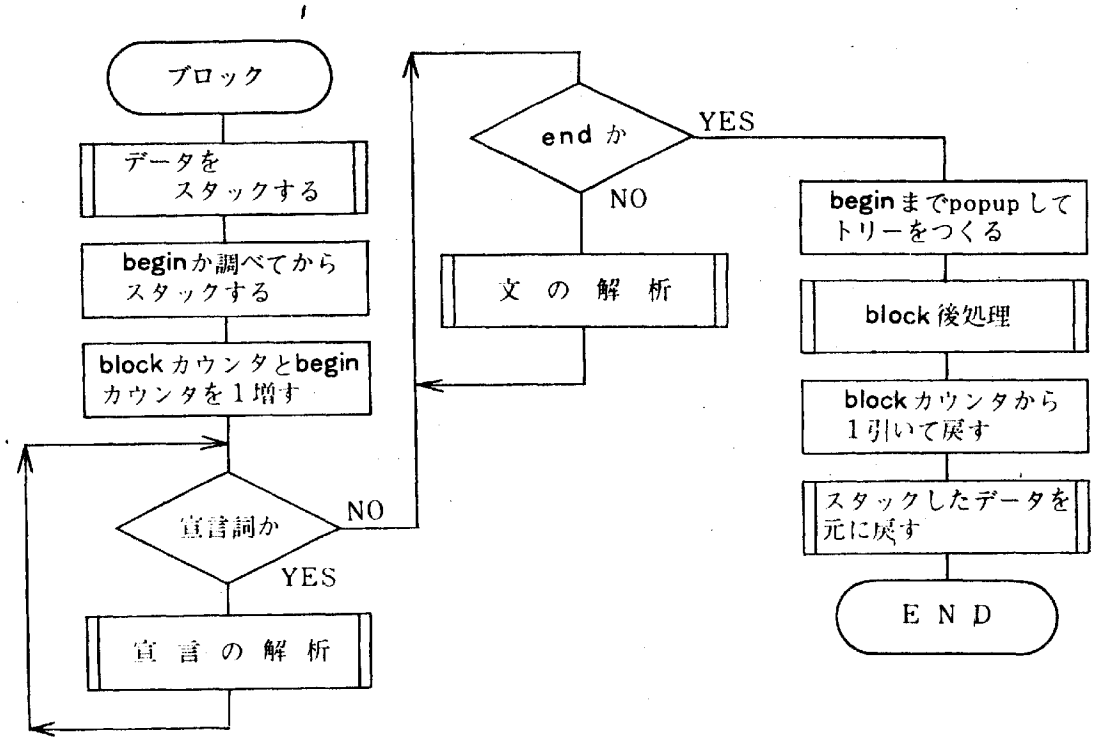


図4 ブロック解析

(7), (8) 表5を参照のこと.

2.6.2. ブロック後処理

ブロック後処理の仕事は，このブロック内で未定義の名札の処理と，このブロックでブロック解析表に登録した名前を抹消して有効範囲を local にすることとの2つである。

未定義で使用された名札は，名札が宣言されることによっても名札の処理のところで確定するが，ブロック後処理で確定するのは次の場合である。自分より外側の親ブロックでは既に定義されていて，自分のブロックでは未定義な場合である。

（例） 例として次のようなプログラムを考える。

```
..... begin .....
  ..... label: .....
    ..... begin .....
      ..... go to label; .....
    ..... end; .....
..... end; .....
```

この例で名札の label が定義されると，ブロック解析表の Z, S, T, X, Y 各パートは o, n, m, l, p となる。但しその時の BLC を n, BGC を m, 名前は label の登録番地を l, 宣言情報表の番地を p とする。更に解析を続けていくと，**go to** label; に出会う。この時このブロックでは名札 label は未定義なので，未定義表が作られる。未定義な名札をもつ **go to** の節の作られた番地を r とすると，この未定義な名札に対する未定義表の Z, S, T, X, L の各パートは 0, n+1, m+1, l, r となる。この時点ではこれ以上処理できないので更に解析を進めると，**end** に出会い，ブロック後処理が行なわれる。ブロック後処理では未定義表の中の S と現 BLC の等しいものを捜す。その未定義表の S パートの値を1下げて，Z, S 及び X をインデックスとしてブロック解析表をサーチする。例の場合は名札の label がすぐ外側の親ブロックで定義されているので見つかることになる。見つけることができたら，未定義表の L パートを参照して，ブロック解析表の Y パートの値を未

定義 go to の節に入れて，その go to の行先が確定する。そして確定した名札を未定義表から抹消する。

しかしこの例はすぐ上のブロックで見つけることのできた場合で，見つからない時は未定義表の S パートの値を 1 引いておくだけである。

第 2 番目の仕事はブロックで使用済みになった名前をブロック解析表から抹消するために Z パートに 1 を入れることである。これはブロック解析表の各要素について，現 BLC と S パートが等しく，且つ Z パートが 1 でないものについては Z パートに 1 を入れればよい。

2.6.3. 文の解析

文の解析の概略は図 5 のようによる。図 5 からもわかるように，第 1，第 2 シラブルでもってどの文であるか判定し，あとは各文の解析ルーチンにコントロールを渡すだけである。文の解析の中にブロックの解析があるので文の解析ルーチン及びブロックの解析ルーチンは回帰的になっている。

(以下次回)

参 考 文 献

[1] 日本規格協会；情報処理 (JIS ハンドブック)，1971.

[2] 森口繁一；ALGŌL 入門，日科技連出版社，1970.

[3] 富士通 K.K.；FACŌM 230-60, ALGŌL 文法編 (SP-051-3-4)，1970.

[4] 沖電気工業 K.K.；ŌKITAC-5090H インストラクションマニュアル，1964.

[5] 岸田孝一；システム・プログラム入門，日本経営出版会，1970.

[6] 田中 一；コンパイラ，森北出版，1971.

[7] 一松 信；近似式，竹内書店，1963.

[8] 広瀬，中田，筧，佐久間，島内；「コンパイラのうちとそと」，*bit*, Vol. 3, No. 1～No. 12，共立出版 (1971).

[9] 穂鷹良介；「Automatic Coding について Ⅲ (1) および (2) ——CŌBŌL Compiler——」，商学討究，第 18 巻 1，2 号，1967.

[10] 穂鷹良介；「Automatic Coding について Ⅳ (1)，(2)，(3) および (4) ——FŌRTRAN Compiler——」，商学討究，第 19 巻 1，2，4 号，第 20 巻 1 号，1968～1969.

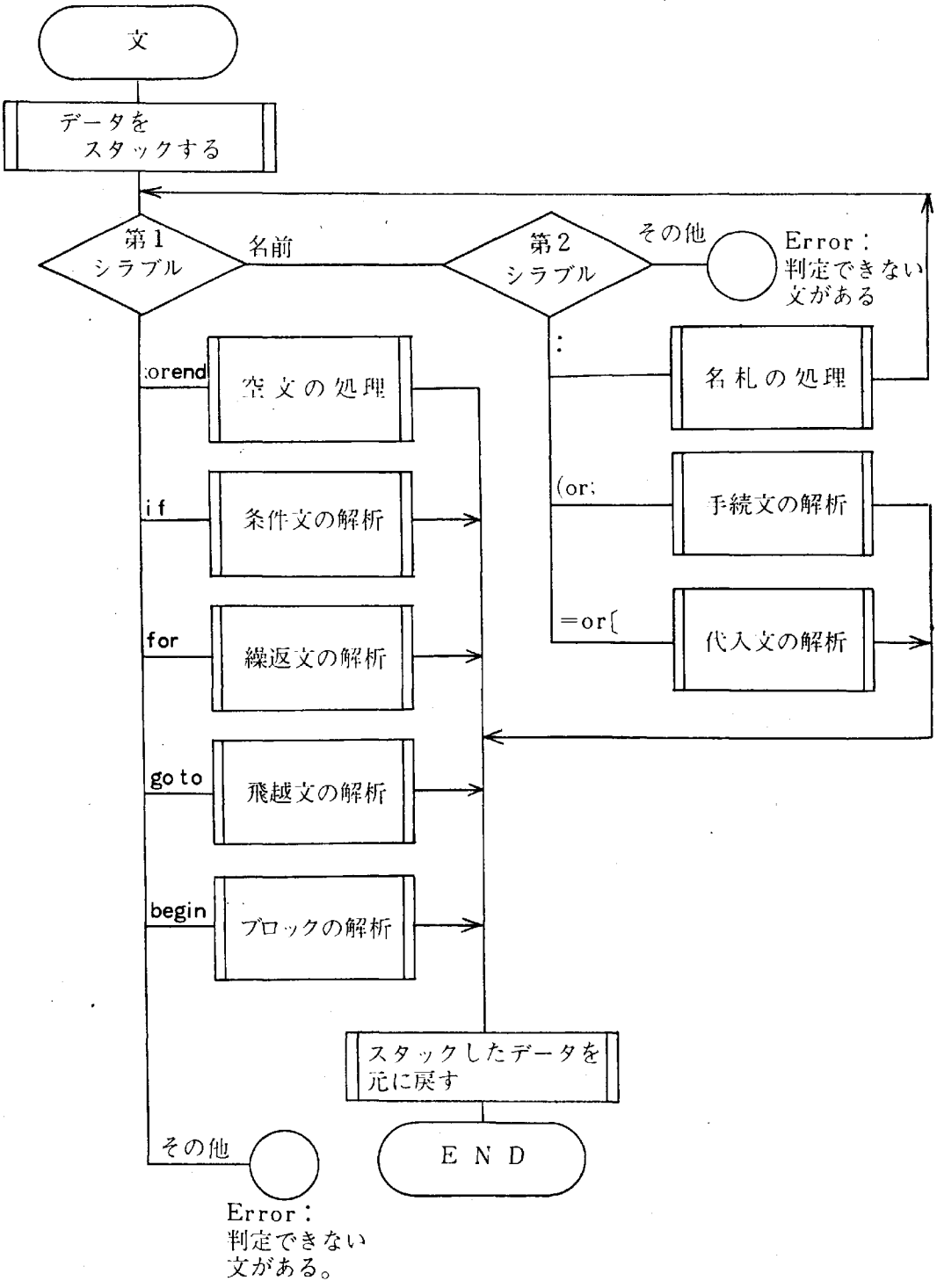


図5 文 の 解 析

(48. 10. 2)