プログラム言語（とコンパイラ）の発展*

和　田　英　一**

プログラム言語の歴史についての最良の記述のひとつは，Saul Rosen によるもの[1]であろう．この種の本の性質上，ごく最近の発展段階は欠けているが，ある時期までのまとめとしては，これまで読んだなかでは十分推奨するに足るもので，未読のかたは是非一読されるのがよいと思う．

そのような優れた記述があるところに，さらにプログラム言語（とコンパイラ）の発展について加筆するようなことはあまりないが，主として筆者の計算機に関する個人的な経験から見た歴史ないし，現状をざっと述べてみたい．

1. 最初のプログラム言語 ——イニシアルオーダ

きわめて初期の計算機，たとえば，Harvard Mk I とか ENIAC とかには，それらの記述を読んでみると，配線盤でプログラムを計算機に教えたようであり，どうもプログラム言語と呼べるようなものは認められない．その次くらいの使い方の計算機がEDSAC, ILLIAC あたりであって，これにはプログラムの「書き方」があって，ひいき目に見れば，この辺がプログラム言語の出発点であるといえよう．EDSAC のプログラム言語は，いまから思いかえせば，アセンブラより遥かに簡単な，しかし，それでもアセンブラ的な，機械語と1：1に対応した言語であった．なにしろ数十語の命令で処理し，格納してゆくのであるから，そんな高級な機能はないのだが，プリセットパラメータが使える，プログラム単位の先頭との相対番地が使える，数値は10進法でいれられるといったようなことができ，当時としては驚くほど画期的なプログラム言語，もしくはプログラム記述方式であった．このプログラム言語は名前はなく，むしろその処理プログラムにイニシアルオーダという名称がついていた（イニシアルオーダの考え方だけでなく，EDSAC のプログラムには，いまでもいろいろ学ぶべきことが少なくない）．

その後 EDSAC ではイニシアルオーダの機能を拡張した実験もおこなわれた．そのひとつが記号番地であり，もうひとつが合成命令である．記号番地はいまのアセンブラではミニコンでさえ常識の機能であり，一方の合成命令はマクロアセンブラの先駆であった．この記号番地の処理に既にリスト構造が用いられていたことは特筆すべきであろう．先年 Wilkes の来日の際，イニシアルオーダの製作者を尋ねたところ，Wheeler の作だという返事であった．

ILLIAC のほうは上述の Wheeler がその後 Illinois 大学に滞在していたときできたので，Wheeler の労作のイニシアルオーダ (DOI) がやはりできたが，このほうは後にドラム記憶装置を使って，記号番地が十分に使えるように改造された．DOI の機能は EDSAC のイニシアルオーダと大体似たものであった．

2. 数式プログラム言語の開幕

最近の電子式卓上計算機とよばれるものは，プログラム（といっても，それが記憶されるものはまだ少ないので，むしろ，定数も含めた演算指令）は数式のとおりにキーを押せばよい，というのが特長の一つになっている．少し高級なほうでは，ミニコンピュータにはよくデスクカルキュレータという電卓シミュレータのプログラムがあるようであり，いずれにしてもプログラムを数式の形で表現したいという希望は，潜在的にあるようだ．

昔パラメトロンができた頃，日本電子測器で製造された PD-1516 は数式プログラムの計算機であったし，同じ頃，東大物理の高橋研にもパラメトロンの数式プログラムのような紙テープまたは鍵盤からの指令で働く計算機があった．

FORTRAN という数式の形で書いたプログラムを機械語に変換する，イニシアルオーダなど，とても及ばないプログラムを IBM で作っているということを耳にしたのは，1958年頃かと思うが，前述の Rosen の本に採録された Backus の報告によると，すでに1954年からプロジェクトが始まっていたようである．

---

* Development of Programming Languages (and Compilers), by Eiiti Wada (Faculty of Engineering, University of Tokyo)

** 東京大学工学部計数工学科

1958年の秋頃に，UNIVAC のエンジニアが FORTRAN のような数式を処理するプログラムがわが社にもある，といっていたのは，後で考えてみると Hopper の Math-Matic のことであったらしい．Rosen によれば，FORTRAN より Math-Matic のほうが古いことになっているが，Math-Matic はどういうわけか FORTRAN のようには流行しなかった．

われわれのところに外国のプログラム言語のニュースが伝わるのは，一昔前はいまほど早くはなくて，たまたま外国へ行ってきた人がニュースを持ってくるのが比較的早いほうであったようである．1959年は Paris で第1回の IFIP の大会があり，そこから帰国した後藤が「英国には AUTOCODE という，数式プログラムの言語がある」と報告したので，そのマニュアルを借りて読んだ記憶がある．

初期の数式プログラムはまだ他にもあったとは思うが，このようなところが一応 ALGOL が登場するまえの主要な言語ないし処理ルーチンであったようだ．これらのコンパイラがどのくらい大変なものかはあまり実感としてつかむことはできなかったが，高橋研ではパラメトロン計算機 PC-1 を使用して，簡単な数式を処理してみていた大学院生もいたように記憶する．

3. アセンブラとオートコーダ

コンパイラの特集号では，イニシアルオーダ同様アセンブラの記述についてはあまり重要でないであろうが，アセンブラ言語にも簡単に触れておきたい．和田にとって，イニシアルオーダのような簡単な入力ルーチンの次に使用できた機械語対応のプログラム処理ルーチンは IBM 650 の SOAP であった．これはアセンブラであり，記号番地を使うことができたが，このアセンブラの特長は IBM 650 がドラム計算機であったため2アドレス方式であり，次の命令の格納場所をどの命令にもつけておくようになっていた（つまり命令がリスト構造になっていた）．次の命令をどこに置くかを，前の命令の演算時間に対して，その間のドラムの回転を考慮して定めるのが，能率のよい目的プログラムとなるわけだが，SOAP はその処理をしてくれるのがその他のアセンブラに見られない最大の特長であった．そういう点では便利であったが，イニシアルオーダのインタールードのようなフレキシブルな機能は反対に使えなくなった．このインタールードはその後のアセンブラでも使えない．

IBM 704 用のアセンブラには FAP とか SCAT とかいうのがあったように思う．650 の SOAP にくらべると計算機も優秀になっただけあって，格段に機能が増した．たとえば，番地部に記号番地や定数の簡単な式がかけるとか，マクロ命令が使えるとか，条件付きアセンブリができるとかである．さらにプログラムをいくつかの部分に分けて，各部分をアセンブラ言語や FORTRAN 言語で自由に書いて，あとで接続できるようになったのもこの頃であったと思う．FAP とは本来 FORTRAN とつなげるためのアセンブラという意味であったらしい．

1961年頃，Illinois 大学の Muller が来日し，当時建設中であった ILLIAC II のプログラム言語について資料を見る機会があった．NICAP とよばれるアセンブラにも番地部に式を書くことができたが，このほうは，実行中に実効番地が，その式のとおりに計算されるプログラムが作り出されるという，いささか変ったアセンブラであった．数式プログラムで，配列要素の番地がダイナミックに計算されるのとほぼ同様な思想である．

アセンブラの一種のようでもあるが，機能が少こし異っているものに IBM 7070（や 1401）で採用していたオートコーダがある．これはアセンブラの類では一番コンパイラに近い．アセンブラと異なるところはどこかといえば，データについて性質を宣言しなければならず，そうすると命令のほうは記号番地からデータの性質をしらべて，たとえ同じ命令の系列を書いてもデータに合った機械命令が作り出される点である．純粋のアセンブラにはデータが一語長か倍長かによって，一語長用の命令が作られたり，倍長用の命令が作られたりするのは，後の IBM System/360 のアセンブラに至るまでない．また，オートコーダではマクロの作り方がマクロジェネレータで，これは他のアセンブラ言語のマクロより PL/I のマクロの機能に近かった．さらにオートコーダには，番地部に数式の書ける ARITH というシステムマクロも用意されていて，主としてこればかり使えばコンパイラと全く同様になってしまうと思われた．

4. ALGOL 60

1960年と1961年に大磯で開かれたプログラミングシンポジウムで，それぞれ森口と清水による ALGOL と ALGOL 60 の報告があった[2,3]．それで遅まきながら和田にも ALGOL が何であるかが少しわかってきて，ACM の報告[4]や淵らによる解説[5]を読んでみ

たりした．ALGOLの制定は，他のこれまでの数式プログラム言語が，どの計算機のためにということで作られたのとちがって，機械とは独立にプログラムを記述するのを目的とした．それは報告からも十分読みとれるのだが，実際計算機との関連を考えると，入出力関係がなかったり，ハードウェアと関係のあるところは適当に省略して書いてあったりして，プログラム言語は計算機をはなれては考えられなかった者にとって，これで果してプログラム言語なのだろうかという疑問があった．

しかし，先見の明のある人たちにより，次々とALGOL 60のコンパイラが作られていた．第2回のプログラミングシンポジウムの報告集にも，すでに「NEAC 2203の Algol Compiler」という報告があり，その後も東大物性研の井上によるPC-2のためのALGOL コンパイラ[6]，清水による OKITAC 5090のための ALGOL/P などが次々と作られ，コンパイラの技術の蓄積がはじまった．

ALGOL 60はプログラム言語の研究のおもしろさを教えてくれたものの一つであった．それ以前に出版されたプログラム言語のマニュアルも，プログラムの書き方とその機能がもちろんはっきり説明してあったが，ALGOL 60の報告では，最初読んだときは何だかよくはわからなかったけれども，その記述が抜群にすっきりとできていた．いうまでもなく，構文のほうはその後のいろいろなプログラム言語の構文の記述に利用された Backus 記法で表現され，大変感心したり，メタ言語の重要性をはじめて認識したりもした．あれだけはっきり構文がかいてあると，コンパイラはかなり自動的にできるのではないかと思ったり，しかし，意味のほうが英語で書いてあるのでは困ると考えてみたりしたものだが，誰もがこんなことを気にしていたらしく，ALGOL 60はやはりその後のコンパイラ・コンパイラの出発点になったように見える．この点に関してはプログラム言語よりはコンパイラ技術になるが，すでに1962年のプログラミングシンポジウムには，この問題について高橋の報告があった[7]．

一体，構文と意味とはどこで分かれるのだろうか，ということがまだ判然としないときもある．ALGOL 60における term, factor, primary のレベル分けによるexpresion の構文の規則は，演算の強さの順が強いほうから↑，*/，＋－であること，また同じ強さの演算は左から右に実行することを示しているという解釈がある[8]（構文1参照）．そうだとすると構文2の decimal number は島内のように0.123をレイテンヒャクニジュウサンと読みたくなるという気持もわかるような気がする．このほかまた ALGOL 60は「副作用」の考えの生みの親である．

⟨adding operator⟩ ::= + | −
⟨multiplying operator⟩ ::= × | / | ÷
⟨primary⟩ ::= ⟨unsigned number⟩ | ⟨variable⟩ | ⟨function designator⟩ | (⟨arithmetic expression⟩)
⟨factor⟩ ::= ⟨primary⟩ | ⟨factor⟩ ↑ ⟨primary⟩
⟨term⟩ ::= ⟨factor⟩ | ⟨term⟩ ⟨multiplying operator⟩ ⟨factor⟩
⟨simple arithmetic expression⟩ ::= ⟨term⟩ | ⟨adding operator⟩ ⟨term⟩ | ⟨simple arithmetic expression⟩ ⟨adding operator⟩ ⟨term⟩
⟨if clause⟩ ::= **if** ⟨Boolean expression⟩ **then**
⟨arithmetic expression⟩ ::= ⟨simple arithmetic expression⟩ | ⟨if clause⟩ ⟨simple arithmetic expression⟩ **else** ⟨arithmetic expression⟩

構　文　1

⟨unsigned integer⟩ ::= ⟨digit⟩ | ⟨unsigned integer⟩ ⟨digit⟩
⟨decimal fraction⟩ ::= . ⟨unsigned integer⟩
⟨decimal number⟩ ::= ⟨unsigned integer⟩ | ⟨decimal fraction⟩ | ⟨unsigned integer⟩ ⟨decimal fraction⟩

構　文　2

5. 記号処理言語

数式による算法言語と共に，記号処理言語も次々と新しいものが作られてきた．IPL-V, FLPL, COMIT, LISP, SLIP, SNOBOL, L^6 などで，これらについてもそれぞれの解説や比較結果が諸々に発表されているから[9,10,11]，ここではことさらにとりあげることはしないけれども，これらの言語の処理はコンパイラであるよりはインタプリタであること，また，そのインタプリタの手法が，コンパイラ作成の技術の基礎として大いに貢献していることなどをまず注意しておこう．ところで，これらの記号処理言語のどれが優れているかはわからないし，あるいはまた，別の記号処理言語を開発しなければならないかも知れないが，この種の言語がマクロの処理に有用であって，マクロ処理を言語処理のレベルから分離するのがよいことは島内が以前から指摘しているところであり，彼の言語設計の基本姿勢のひとつである．この意見は十分傾聴すべきものと思われ，記号処理言語が将来図形処理への応用とともに広く用いられるようになる分野と期待してよいであろう．

6. PL/I

FORTRAN, ALGOL, COBOL などそれぞれ特長

をもった言語が出揃うと，それらの機能をすべておおうような言語を作りたくなるのが人情なのであろうか．PL/I はそういった先輩格の言語の機能をカバーし，さらにそれらの言語ではできなかったことも可能にすべく（たとえば，コアのダイナミックアロケーションやコンパイル時のプログラム発生機能など），1963年頃に IBM のユーザの会であった SHARE で設計された．この辺の事情や PL/I と FORTRAN, ALGOL, COBOL との機能の比較は竹下の解説に詳しいのでそれを参照されたい[12,13]．また，PL/I でリスト処理を行なうための手法の提案なども早くからあって[14]，この言語の強力なことを実証しているが，反面やはり初心者にはむずかしいほうの言語になってしまったようである．PL/I は上述のように一般のユーザが，問題解決のためにプログラムを書くときに利用されるほか，各所でシステム記述用言語として利用されていることも注目しておいてよいであろう[15,16]．MIT の Multics の EPL (Early PL)，日立製作所中研のタイムシェアリングシステムの PL/IW は，それぞれフル PL/I からシステム記述用に適した部分だけを抽出したものといわれている．この他にもシステム記述に既製のプログラム言語が利用されたこともあるが（たとえば，東大亀田の TSS の FORTRAN による記述[17]），ここで PL/I が採用された理由を考えてみると，

（1） システム記述用の言語を考えてみると，いろいろとり入れてみても，各種の機能を持った PL/I のサブセットになってしまう．

（2） 将来どの計算機でも PL/I は総合的な言語だということで，PL/I のコンパイラが作製されるようだと，PL/I で書いておくのが最も将来性がある．

というようなことであるらしい．

7. PUFFT, QUICKTRAN, BASIC, JOSS など

オペレーティングシステムの進歩とともにプログラム言語も多少影響をうける．この節ではそういう星の下に生まれた言語をいくつかひろってみることにする．

まず PUFFT[18] は Programming Systems and Languages の編者 S. Rosen らの作で，これは名前の示すとおり FORTRAN である．FORTRAN も大形高速な計算機のオペレーティングシステムの制御下では，システムモニタにより FORTRAN コンパイラが何度も補助記憶装置から磁気コアに読み出され，いくつかのプログラム単位の処理が行なわれ，最後にリンケージエディタによりエディット，ロードされ，目的プログラムが走る段階に到達する．記憶場所を十分とり，一旦ロードされると比較的長時間動いている目的プログラムなら，この種のオペレーティングシステムとコンパイラの構成でもよいが，学生やその他の初心者が記憶場所も少なく，計算時間も短いプログラムをかけている場合，また，そういうジョブが連続している場合は，上述のオペレーティングシステムでは，計算機は絶えず補助記憶装置のプログラムを磁気コアのほうへ転送しているだけで，計算機の効率は大変悪いことになる．そこで PUFFT は磁気コアの部分に常駐する，小さい目的プログラムのためのコンパイラとして設計された．このコンパイラは記述を読むと実に巧妙にできていて感心する．それに報告だけではあまりよくわからないが，エラーメッセージの構成がまた特に良くできていて，親切に打ち出されるともいう．とにかくオペレーティングシステムにあまり世話にならないから，非常に速くて，報告よれば IBM 7090 のモニターと Version 13 とよばれる高速化された FORTRAN コンパイラで 20～30 秒かかる小さいジョブが，PUFFT によれば 2～4 秒ですむのだそうだ．この成功をみて Waterloo 大学でも同様の高速コンパイラを作ったそうで，その報告も何かに出ていたと思う．Edinburgh 大学の Michaelson は，数少ない成功したソフトウェアのひとつがこの PUFFT だという[19]．

タイムシェアリングシステムが登場するとタイムシェアリング用の FORTRAN ともいうべき QUICKTRAN が出現した．これはもうずいぶん昔読んだので，また資料も手元に見つからないので記憶を辿ってのべることになるが，一応インタプリタ方式であり，端末からソースプログラムが入力されると，それはコンパイラでインタプリタ用の中間言語に翻訳されてユーザのファイルに登録される．つまりソースプログラムはファイルには残らない．それが時分割でよび出されてはインタプリタで間接実施されるというものであった．おもしろいのは，この QUICKTRAN にはデコンパイラがあって，端末にいるユーザがソースプログラムを見たいと指示すると，デコンパイラが中間言語をまたソースプログラムの形に戻して送りかえすので，ソースプログラムに冗長な括弧でもあると，戻ってきたプログラムにはそれがなかったり，式の書き順も違ってくることがあったりする言語だったように

思う．

これはコンパイル技術に関連するが，タイムシェアリングでは，インクレメンタルにコンパイルしたいという希望が一方にあり，他方には以下のコンパイルに大影響を与えるようなステートメントが変更されて，コンパイルを全部やり直さなければならない現実の問題があり，どのようにするのがタイムシェアリングシステムでのプログラム言語はよいのか，和田にはまだ全然わからない．この問題でのひとつの試みは牛島らによって九大で行なわれた[20]．

タイムシェアリングシステムで計算機への接近が容易になると，FORTRAN よりも簡単な言語を開発しようという動きが見られるようになり，その一番打者が Dartmouth 大学の Kemeny らによる BASIC である．これは大ざっぱには FORTRAN だが，機能を極力減らし，さらに書き方も変更してしまったので，FORTRAN と似ていなくなってしまった．この言語はタイムシェアリングシステムのコマンドとレベルが混合しているようで，BASIC の文法書を読んでいるのかタイムシェアリングシステムのコマンドの説明書を読んでいるのかわからなくなるくらいである．そうまでしなくてもインクレメンタルな実行もできそうに思うがいかがであろうか．

また FORTRAN ではいけなかったのだろうかと疑問になる（FORTRAN は代入文がキーワードなしではじまり，コマンドを待ち受けているシステムにとってはあまり嬉しくないものだが，BASIC は必ずステートメント番号をもっているので，コマンドとステートメントとの区別は容易につく，また一意名の作り方も簡単なので，キーワードとの区別は容易である）．

Dartmouth の TSS には ALGOL 60 もあり，この方はコマンドとは別レベルらしい．

タイムシェアリングというよりは対話しながら計算機を使う方式の言語のひとつは JOSS とその類似言語である（たとえば英国 Cambridge 大の FIGARO[21]）．これは BASIC が FORTRAN 的にプログラムをいれてから開始するのに対し遥かに会話的で，タイプするかたわらから答が打ち出される．しかし，式を記憶させておくこともできるし，複数行の式からなるサブプログラムを記憶させて，後からそれを実行することもできるようになっている．BASIC とくらべると JOSS とその類似言語のほうが実用的であるように思える．

FIGARO はその使い方の指示まで端末に打ち出されるので，説明書もなしに黙って坐ればすぐタイムシェアリングで計算機が使えるようになるというシステムであるらしい．

8. ALGOL 68

1960 年に ALGOL 60 が発表され，1963 年頃にその改訂版が出てからも ALGOL に対する作業は進行していた．その辺の事情は ALGOL 68 の序文[22]やその他[23,24,25]を参照されたい．ALGOL 68 としてはオランダの Mathematisch Centrum の Wijngaarden 達から 1967 年 11 月に “A Draft Proposal for the Algorithmic Language ALGOL 68” が MR 92 として出版され，つづいて翌年 1 月に MR 93 が出版された．MR 93 は Algol Bulletine の予約者にも配布されている．しかし 1968 年春の WG 2.1 では MR 93 は ALGOL 68 として認められず，したがって Wijngaarden 達は 10 月に改訂版 MR 99 を世に問うた．この MR 99 は WG 2.1 のメンバーによって検討された結果，1968 年もおしつまった 12 月末に München の WG 2.1 の会合において作業グループのレベルで ALGOL 68 として承認されるに至った．その後 IFIP の総会でも認められ正式に ALGOL 68 となった．最終的な報告書は MR 101 と称されるもので，それも 10 月版が誤りが少ない．Numerische Mathematik[26] にも発表されている．

ところで，報告書によると ALGOL 68 は ALGOL 60 にくらべるとその拡張にあらずして，むしろ完全に新しい言語であり，したがって ALGOL 60 にくらべどう差異があるかを簡単にのべることはできないが，次の点で異なるとしている．すなわち

i）ALGOL 60 の 3 種のタイプに対して無限種のモード．

ii）基本のモードは種々の長さをとりうる実数と整数，それに Boolean と文字．

iii）ALGOL 60 の配列に対して，マルティプルと構造（ストラクチャ）．

iv）名前という値の導入．ある値に名前をつけると，その値をその名前が「さしている」値といい，名前の値と区別する．

v）ルーチン（手続き本体）もデータとして取扱う．また書式の概念．

vi）モード宣言により，すでに宣言してあるモードから新しいモードが作れる．いくつかのモードの混合のモードが作れる．

vii) 優先度の宣言とオペレーションの宣言があって，新しい（二項）演算を導入することができる．

viii) 並行処理（コラテラル）ができる．

ix) 標準の宣言により，狭義の ALGOL 68 にない機能を一般の利用者のために追加できる．

などである．この線に沿って ALGOL 68 が設計されているが，ALGOL 60 なら意味として記述してあるような部分も，できるだけ構文の範囲にいれようとした努力のあとが見られる．

構文の記述は ALGOL 60 にくらべるとはるかに複雑であって，生成規則において ALGOL 60 でメタ変数といっていたものをノーションとよび，そのノーションをさらに生成する一段上のメタの生成規則が用意されている．この一段上のノーションはメタノーションといわれ，この生成規則ではノーション，またはノーションを綴っている文字がターミナルになっている．この辺の機構を示す最も簡単な例をさがしてみると，構文3のようなものがある．

```
NOTION option: NOTION; EMPTY.
```

構 文 3

これは NOTION option というノーションは，NOTION というノーションが EMPTY というノーションかからなる，を意味している．ところで NOTION とか EMPTY とか大文字で書いてある部分はメタノーションで，これはメタ生成規則を見なければわからない．この部分に関係あるメタ生成規則は構文4のも

```
NOTION: ALPHA; NOTION ALPHA.
ALPHA: a; b; c; d; e; f; g; h; i; j; k; l; m; n;
  o; p; q; r; s; t; u; v; w; x; y; z.
EMPTY:.
```

構 文 4

のである．すなわちメタノーション NOTION はメタノーション ALPHA かメタノーション NOTION のうしろにメタノーション ALPHA がついたものである．一方，ALPHA はノーションを綴るための小文字のいずれかであるから，メタノーション NOTION は a とか b とか…いうものである．仮に a をとると構文3から「a option は a か EMPTY の形」ができる．メタノーション EMPTY はメタ生成規則ではいきなり点がきているから結局

a option: a;.

が得られる．同様にしてこれから b option: b;. や ab option: ab;.…が作られる．どこまで作っておけばよいかは知らないが，これで，たとえば別の生成規則の中に label sequence option というノーションがあらわれると，これを label sequence におきかえた構文と，これを除去した構文が作り出されるという仕掛けである．

ALGOL 60 にくらべて断然難解になっているので，別に解説書が出るそうである．上述の説明にも正確でないところがあれば容赦していただきたい．

9. ALGOL N

MR 93 で ALGOL 68 の提案を知った情報処理学会の ALGOL 作業グループ AWG と，ランゲージ記述グループ LDG の有志では，それよりももっとわかりよい ALGOL 案を作って，WG 2.1 に提出しようということになり，同年夏から12月にかけて何度か会合を開き，ひとつの案をまとめた[27]．それが ALGOL N（第1版）であり，N はニッポンのつもりであった．この案は12月に München で開かれた WG 2.1 に米田によって提出された．第1版は急いで作ったため不満足な点も少なくなく，目下第2版の作成中であるが，第1版の ALGOL N の処理系は京大の佐久間[28]により作成されている．ALGOL N は ALGOL 68 を意識しているので ALGOL 68 でできることは何らかの形でなるべく N でもできるようにと考えたほか，総体に（構文もエラボレーションも）ALGOL 68 よりずっとわかり良いようにしたこと，ALGOL 68 のような型（68 ではモード，N ではタイプ）の自動的な変換（コアーション）を廃止し，型の混合の型もやめた．新しい演算の定義は 68 同様自由にできるようにしたが，これは加減乗除のような普通に演算と思われているもののほか，代入や if, then, else も式の中の演算という扱いにしてしまった．これらの演算は狭義の ALGOL N 中に始めから用意されているのではなく，68 のように標準の宣言中でおこなわれている．手続きもデータであること，構造のデータのあること，他の値の場所をさしているものもデータとする点なども 68 と似ているが，最後の場所をさしているデータはそこに入っているデータの型には影響をうけないところが ALGOL 68 とは異っている（ALGOL 68 では実数をさす，とか実数をさすものをさす，とかがそれぞれモードだが，ALGOL N では，なにかを「さす」のタイプしかない）．

改良中であるが第2版の現在の構文案を構文5に示す．構文記法のうち〈,〉に挟まれているのはメタ変数，＝は「の形」で BNF の ::= である．縦棒，形

```
<expression>    = <secondary>
                | <formula>

<secondary>     = <primary>
                | <procedure notation>
                | <array element>
                | <structure element>
                | <procedure call>

<primary>       = <identifier>
                | <go to statement>
                | <block>
                | <closed expression>
                | <code>
                | <effect notation>
                | <real notation>
                | <bits notation>
                | <string notation>
                | <reference notation>
                | <array notation>
                | <structure notation>

<declaration>   = <variable declaration>
                | <formula declaration>
                | <mark declaration>
```

```
<real donor>      = [ <number> ]
<bits donor>      = [ <bits> ]
<string donor>    = [ <string> ]
<array donor>     = ( <expression> {,} ooo )
<structure donor> = ( [(<selector> <expression> ) {,} ooo ] )
<procedure donor> = [ : ( [<identifier> {,} ooo] ) <primary> ]
<modifier>        = [ ( [<expression>] ) ]
<array bound>     = [ [<expression> :][<expression>] ]
<frame>           = [ ( ) ] <mark> [ ( ) ] {<mark>} ooo
<left priority>   = [before ([<mark> {,} ooo] | all) left]
<right priority>  = [after ([<mark> {,} ooo] | all) right]
<identifier>      = <letter> [ <letter> | <digit> ] ooo
<selector>        = ( <letter> | <digit> ) ooo :
<number>          = <digit> ooo / . <digit> ooo / ( 10+ | 10- ) <digit> ooo
<bits>            = ( 0 | 1 ) ooo
<string>          = ' [ <character> ] ooo '
<basic symbol>    = <non comment> | comment | silent
<non comment>     = <letter> | <digit> | <delimiter> | <donor symbol> |
                    <code body>
```

<go to statement> = go to <identifier>

<block> = begin [<declaration> ;] ooo
([<identifier> :] ooo <expression>) {;} ooo end

<closed expression> = (<expression>)

<code> = code <structure donor> <primary> : <code body>

<effect notation> = effect

<real notation> = real <modifier> <real donor>

<bits notation> = bits <modifier> <bits donor>

<string notation> = string <modifier> <string donor>

<reference notation> = reference

<array notaion> = array <array bound> < array donor>

<structure notation> = structure <structure donor>

<procedure notation> = procedure ([<expression> {,} ooo])
<primary> <procedure donor>

<array element> = <secondary> [<expression>]

<structure element> = <secondary> [<selector>]

<procedure call> = <secondary> ([<expression> {,} ooo])

<formula> = [<expression>] <mark> [<expression>] {<mark>} ooo

<variable declaration> = let <identifier> be <expression>

<formula declaration> = let <frame> represent <expression>

<mark declaration> = let <mark> operate <left priority><right priority>

<letter> = a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|
x|y|z

<digit> = 0|1|2|3|4|5|6|7|8|9

<delimiter> = <straight symbol> | <extension symbol> | <mark>

<donor symbol> = .|$_{10}$+|$_{10}$-|0|1|'|'|<character>

<straight symbol> = begin|end|before|left|after|right|(|)|[|]|
{|}|[|]|effect|real|bits|string|reference|array|
structure|procedure|let|be|represent|operate|
code|go to|all|,|;|:|:

<extension symbol> = integer|Boolean|character|name|quantity

<mark> = nil|dummy|type|copy|new|enproc|enref|deref|match|
as|if|then|else|←|:=|≡|≠|≠|π|e|<|≤|>|≥|+|-|
×|$^{-1}$|/|↑|↑|entier|round|float|sign|abs|arg|√|exp|
log|$\log_{10}$|sin|cos|$\tan^{-1}$|mod|true|false|∧|∧|filler|
size|lower bound|upper bound|conc|*|from|up to|
at|set|find|in|replacing|first|with|¬|∧|∨|⊕|
complex|i|Re|Im|$^-$|⊥|.|list|car|cdr|for|:=|do|step|
until|while|case|of|collateral|constant|scale|
precision|exact|varying|mode|<etc 1>

<character> = â|b̂|ĉ|d̂|ê|f̂|ĝ|ĥ|î|ĵ|k̂|l̂|m̂|n̂|ô|p̂|q̂|r̂|ŝ|t̂|û|v̂|ŵ|x̂|ŷ|ẑ|
0̂|1̂|2̂|3̂|4̂|5̂|6̂|7̂|8̂|9̂|(̂|)̂|+̂|-̂|.̂|,̂|"̂|^̂|→̂|⇒̂|⇒̂|<etc 2>

<comment> = comment [<non comment>] ooo silent

< etc 1>, <etc 2> and <code body> are unspecified.

構 文 5

の大きい大，中，小の括孤，大きい斜め線，それに。。。が構文生成規則用のメタ言語の記号で，それ以外がメタ定数ということになる．小括孤は式の優先関係を示す．中括孤はその中はあってもなくてもよいの意．。。。はその左のものの1回以上のくりかえし，。。。のすぐ左に大括孤ではさまれたものがあれば，それをセパレータにして，その左のものの1回以上のくりかえしを表わす．斜め線はその左右どちらかだけをとってもよし，また両者をコンカテネートしてとってもよい記号で，縦棒はBNF同様「または」の記号である．操作の強さの順は。。。が最強，つぎがコンカテネーション，弱いほうが縦棒か斜め線である．

ALGOL Nの設計思想のひとつは，PL/Iのようなはじめからいろいろな型のデータでも扱える単一の言語というのではなく，適当な標準の宣言を付加すればそれでALGOL 60のようにも，LISPのようにも，また図形処理や論理回路設計用言語にでも，任意の言語として一般の利用者に使ってもらえるようになる基礎の言語を作っておくことであった（ハードウェアのマイクロプログラムの方式設計に似ている）．したがってALGOL Nはいろいろな方面の言語のどれもの基本となっているような機能だけを準備し，あとは利用者ないし標準の宣言の作成者が目的に応じて拡張して使うことになっている．もっとも，それだけではALGOLとはいえないから，最低ALGOL 60の程度の標準の宣言は，基本的な標準の宣言としてALGOL Nの作成メンバーで作り，付録としてつけておくことになっている．これが気に入らない利用者はもちろん除去して自分の標準の宣言をつけてALGOL Nを使って差しつかえない．

標準の宣言の部分を，コンパイラでどう扱うかがALGOL Nのコンパイラ作りの問題点のひとつである．標準の宣言は手続きの形でおこなわれるから，それをサブルーチンとして一々そこへジャンプすることも考えられるが，たとえばa:=a+1;でまず+のサブルーチンへとび，ついで:=のサブルーチンへとぶようにコンパイルしたのでは全く実用にならない．そこで標準の宣言中の手続きの内容を解釈してコンパイルされたプログラムに直接組み込みたいのだが，そんなにうまく行くかどうかわからない．うまくゆけばコンパイラ・コンパイラができたようなものだが，これは今後のALGOL Nの最も重要な研究課題となるであろう．

10. コンパイラについてのコメント

コンパイラの発展については，あまり報告する材料を持ち合わせないが，ALGOL 60が登場したころから今まで，人間がせっせとコンパイラを作っているという状態と，コンパイラを自動的に作ろうという努力とが定常的に続いているように思われる．

Californiaの大学のHuskyによるNELIACコンパイラの講演会が東京で開かれたのはもう何年前になるであろうか．とにかく算術式の分析をオペレータの強さのマトリックスを用いてうまくできることを説明していた．その後プログラミングシンポジウムで池野によるAUTOCODEの算術式の分析の発表もあり，CACMには中田よりHARP 5020の技法の一部が報告され[29]，またコンパイラのシンポジウムが2回ほど開催されたりもしたが[30]，あまりコンパイラの技術に関する報告は国内では多くないようであった．しかし，現実には次々と新機種のためのALGOLやFORTRANのコンパイラがメーカーやソフトウェア会社によって作られているのだから，技術は段々と蓄積されているはずである．国外のものでは1960年にACMのCompilerシンポジウムがあり[31]，またその後のいろいろな文献はRosenの本[1]などに見られる．

一方，コンパイラの自動作成については最近井上の報告があり[32]，またFeldmanの労作もあって[33]，その概況を知る資料は事欠かない．この方面の詳細はこの2編に委ねることとする．さらに昨年夏コンパイラの自動作成に関するシンポジウムが開催されたことも記憶に新しい[34]．

参考文献

1) S. Rosen (editor): Programming Systems and Languages, McGraw-Hill, 1967.
2) 森口繁一：ALGOLの文法，第1回プログラミングシンポジウム報告集.
3) 清水留三郎：算法言語 ALGOL 60，第2回プログラミングシンポジウム報告集.
4) P. Naur ほか：Revised Report on the Algorithmic Language-Algol 60, Programming Systems and Languages.
5) 淵一博ほか：算法言語 ALGOL 60に関する報告，情報処理，Vol. 1, No. 3 (1960, Nov.) p. 157.
6) 井上謙蔵ほか：ISSP ALGOLのコンパイラ，情報処理，Vol. 5, No. 1 (1964, Jan.) p. 9.
7) 高橋秀俊：自動プログラミングにおけるSyntax Analysis，第3回プログラミングシ

ンポジウム報告集.
8) 長尾真: Phrase Structure Language のプログラミング言語への応用, 昭和42年電気四学会連合大会シンポジウム Phrase Structure Language.
9) J. B. Bobrow (editor): Symbol Manipulation Languages and Techniques, North-Holland, 1968.
10) D. G. Bobrow, B. Raphael: A Comparison of List Processing Computer Languages, Programming Systems and Languages.
11) 黒沢俊雄ほか: 数式処理, 情報処理, Vol. 10, No. 2, (1969, March) p. 78.
12) 竹下享: 新しいプログラミング言語 PL/I の出現とその特徴, 情報処理, Vol. 7, No. 4 (1966, July) p. 202.
13) COBOL 研究会: 竹下氏の PL/I の解説について, 情報処理, Vol. 8, No. 2 (1967, March) p. 88.
14) H. W. Lawson Jr.: PL/I List Processing, CACM, Vol. 10, No. 6 (1967, June) p. 358.
15) P. G. Newmann: The Role of Motherhood in the Pop Art of System Programming, ACM 2nd Symposium on Operating Systems Principle, 1969.
16) 浜田穂積ほか: PL/IW によるシステムの開発, 第11回プログラミングシンポジウム報告集.
17) 高橋秀俊, 亀田寿夫: オペレーティングシステムの一構成法, 情報処理, Vol. 11, No. 1 (1970, Jan.) p. 20.
18) S. Rosen, R. A. Spurgeon, J. K. Donnelly: PUFFT-The Purdue University Fast FORTRAN Translater, Programming Systems and Languages.
19) S. Michaelson: How to Succeed in Software, IFIP, 1968.
20) 牛島和夫ほか: 構造をもった言語に対する会話的プロセッサ, 情報処理, Vol. 9, No. 1 (1968, Jan.) p. 7.
21) M. V. Wilkes: Time-Sharing Computer Systems, McDonald, 1968.
22) A. van Wijngaarden (editor) ほか: Report on the Algorithmic Language-Algol 68, Second Printing by the Mathematisch Centrum, Amsterdam, MR 101, 1969, Oct.
23) 長尾 真: ALGOL language の最近の動向, 情報処理, Vol. 3, No. 4 (1962, July) p. 181.
24) 井上謙蔵: ALGOL の動向, 第6回プログラミングシンポジウム報告集
25) 森口繁一: 新しい ALGOL について, 情報処理月例会資料 3, 1965-6-15.
26) A. van Wijngaarden (editor) ほか: Report on the Algorithmic Language-Algol 68, Numerische Mathematik, Band 14 (1969) p. 79.
27) S. Igarashi ほか: ALGOL-N, Algol Bulletine, No. 30 (1969).
28) 作久間紘一: ALGOL-N Compiler 作成について, 第11回プログラミングシンポジウム報告集.
29) I. Nakata: On Compiling Algorithm for Arithmetic Expression, CACM, Vol. 10, No. 8 (1967, Aug.) p. 492.
30) 山本欣子: ある Symposium より, 情報処理, Vol. 7, No. 3 (1966, May) p. 164.
31) ACM: ACM Compiler Symposium, CACM. Vol. 4, No. 1 (1961, Jan.) p. 3.
32) 井上謙蔵: ソフトウェアの自動作成, 情報処理, Vol. 10, No. 4 (1969, July) p. 191.
33) J. Feldman, D. Gries: Translator Writing Systems, CACM, Vol. 11, No. 2 (1968, Feb.) p. 77.
34) コンパイラ自動作成シンポジウム委員会: コンパイラ自動作成シンポジウム報告集, 1969.

(昭和45年4月27日受付)