コンピュータサイエンスシリーズ

コンパイラ・コンパイラ

コンパイラ・コンパイラ

井上謙蔵著

産業図書

＜著者略歴＞

井上謙蔵（いのうえけんぞう）

昭和20年　九州大学理学部卒業
昭和22年　東京大学理工学研究所助手
昭和32年　東京大学物性研究所助手
昭和36年　東京大学物性研究所講師
昭和41年　富士通入社，部長付
昭和48年　東京工業大学理学部教授，工学博士

コンパイラ・コンパイラ　　定価 2300 円

昭和 45 年 11 月 12 日　初　版
昭和 52 年 4 月 5 日　第 4 刷

著　者　井　上　謙　蔵

発行者　森　田　勝　久

発行所　産業図書株式会社
東京都千代田区外神田 1-4-21
郵便番号 101-91
電話東京 (253) 7821 (代)
振替口座東京 2-27724 番

横山印刷・関口製本

序

コンピュータとその応用の急激な発達は，巨大なオペレーティング・システムを生みだし，利用者にも製作側にも，その大きさが重圧となりはじめている．この巨大化の傾向は，現在のところとどまる様子をみせていないばかりか，応用分野の拡大とともに，もっと促進される方向にある．

コンピュータの現在の利用方法は，ようやく手廻し計算機的な活用の極限に達したようにみえる．もちろん，このような用法は今後も，１つの重要な分野であろうが，将来は，もっと知脳的な面での活用が日常茶飯事になり，あるいは自己学習的な機能がオペレーティング・システムに要求されるようになるかもしれない．

このような時点で，ソフトウエアとの総合システムとみるとき，コンピュータは，なお発展の前期にあって，はやくもオペレーティング・システムの巨大化によって，システムの機能の喪失を暗示する危機的症状を示すにいたった．その様子は，都市の巨大化と，そのための都市機能の喪失に，よく類似している．

われわれは，オペレーティング・システムの機能と構成を再評価し，将来の方向に対するするどい洞察の上に，いわゆるソフトウエア危機を脱出する方法を発見しなければならない．そのためには論理的な手段が要求されるが，オペレーティング・システムに対して，われわれはそれをほとんど欠如しているといっても過言ではあるまい．

オペレーティング・システムの中にあって，コンパイラのみは，製作の自動化を目標に，論理的基盤の上に，構造の再編成がはかられている．オペレーティング・システム全般と，コンパイラとでは，機能と構造の論理は，まったく異なるであろう．しかし，コンパイラに対する方法が，論理化という点で，１つの示唆を与えることは確かであろう．

コンパイラ作製の自動化の中心的方法は，コンパイラ・コンパイラの方法である．本書は，コンパイラ・コンパイラの機能と構造の基本的部分の解説を目標とする．

コンパイラ・コンパイラは，コンパイラ技術の形式化の上に成立するものである．そこ

で解説の中心は，プログラム用言語の文法の形式化と，その解析方法との一般的な関係におかれる．プログラム用言語の文法は，文章の組み立て方，すなわち構文の規則と，それに対する意味づけよりなるが，現在研究の進んでいるのは前者であって，後者の形式化の問題は，解決の方向があまり明白でない．そこで，本書の解説も，主として構文の規則と，文の解析の問題に集中される．

第1章において，ソフトウエア危機の実態を，もう少しくわしく論じ，第2章でコンパイラ作製を簡易化する諸方法を概観する．

第3章では，コンパイラ・コンパイラ導入の最初のきずなとなった ALGOL 60 の文法記述法についてのべる．これは，まず文法の概念に馴れるためでもある．第4章では，その一般化として，生成文法の一般論にふれ，その1つの類，文脈独立文法と，それに対する構文解析法を論ずる．

第5章で，実用的観点から，文脈独立文法の下位の類である順位文法と限定文脈文法，およびそれらの変種，拡張について，文法と解析法との関連を示す．これらの解析法は，第4章のものが未確定的であるに対し，確定的解析手段を与える．

第6章では，実例によって意味づけの方法をのべる．

第7章では，コンパイラ・コンパイラの機能と構成をなるべく一般的に論じ，その構文解析部発生ブロックの詳細な構造を第8章に示す．

以下，本書で言語というときはプログラム用言語を意味する．また特に，第4章と第5章では，生成文法に関連して，言語の構文のみを問題として言語と呼ぶ．したがって，文法というときは，前者では意味づけを含んだ呼称であるし，後者では意味づけには何の関係もない．

本書は，コンパイラ・コンパイラに関係をもつすべての知識を網羅するものではない．しかし，文法の記述法について，新しい方向をはらんでいる ALGOL 68 や ALGOL N などとの関連を割愛したのは残念である．

ソフトウエア危機の克服という見地から，本書がコンパイラ・コンパイラの論理と技術の現段階について，何がしかの展望を読者に与えうるなら，著者の幸いとするところである．

昭和 45 年 10 月

著　者

目　　次

1. 緒　　論

1946年にプログラム内蔵方式の最初のコンピュータ，EDVAC の建設がペンシルバニア大学ではじまった．これは完成されなかったが，同じ原理のもとに設計された EDSAC が，1947 年にケンブリッジ大学で稼動にはいった．それからわずか 20 年の間に，数百の端末利用者が，同時に1つの巨大なコンピュータを遠隔地より使用できる，いわゆるタイム・シェアリング・システムや，大陸に散在する計算センターを1つに結びつけるコンピュータ網が出現する段階に達した．

ところで，このような巨大なコンピュータ系はもちろんのこと，1つの計算センターに自立的に存在するコンピュータに関しても，現在ではその運転のために膨大なソフトウエアの体系が必要とされる．それは，コンピュータの速度が EDSAC の当時にくらべ 1000 倍以上もはやくなったことと，コンピュータの利用者が限られた専門家の集団から，さまざまな職業の一般大衆へ広がったことに対処するためである．このような発展の結果，生じた問題は，つぎの点に要約される．

（1） コンピュータ本体と入出力装置の速度の断層

コンピュータの本体の速度は非常にはやくなったけれども，入出力装置の速度はそれに追いつかず，データの入力や，演算結果の出力時間を含めたプログラム実行の総合的な時間は，もっぱら，入出力を待ち合わせる時間に支配されるようになった．入出力は人間と直接のかかわりあいをもつし，カードや紙送りなどの機構をもつので，無制限にはやくすることもできないという事情もある．これでは，コンピュータの速度をいくらはやくしても，むだであるから，コンピュータの演算と入出力を併行に進めることが必要である．その結果，入出力のプログラム上の取扱い方は大変むずかしくなるので，特殊なソフトウエアの力を借りなければならなくなった．

（2） プログラム用言語と機械語のずれ

FORTRAN, ALGOL, および COBOL 等々の問題向き言語の発展によって，コン

ピュータの専門家ではないが，それぞれの分野での専門家が，容易にプログラムを組むことができるようになった．しかし，これらの問題向き言語と，コンピュータに固有の機械語との間には，大きなずれがあるため，問題向き言語によるプログラムは，機械語に翻訳されない限り，コンピュータを動かすプログラムにはならない．この翻訳のために，コンパイラ（compiler）と称するプログラムがつくられる．またプログラムを組みやすくするためには，ひんぱんに使われるサブ・ルーチン，たとえば三角関数のような，いわゆるライブラリは，すべてコンピュータの記憶装置に内蔵されていて，名前だけで引用できるようにしなければならない．

（3） プログラム実行にいたる手続き

1つのプログラムの実行が終わったあと，つぎのプログラムをコンピュータにのせるために，カードをカード読取装置にのせるとか，磁気テープを磁気テープ装置にかけるとか，いろいろな作業が必要である．これらをすべて人手にたよっていたのでは，コンピュータの本体からみれば数分の遊び時間ができる．1分，何万円という現代のコンピュータでは，これは膨大な損失となるし，(1) と同様な理由で，コンピュータの高速化の効果を減殺する．それゆえ，プログラムの運行は，何らかの手段で，自動的につかさどられなければならない．

（4） 高価なコンピュータを有効に使うために，記憶装置や入出力装置なども，むだなく使うような工夫がなされなければならない．

これらの問題は，現在モニタ（monitor）と称するプログラムに監督されながら動く，アセンブラ（assembler），コンパイラ，データ管理プログラム，エディタ（editor），ローダ（loader），ユティリティ（utility），その他のライブラリからなる，オペレーティング・システム（operating system）によって解決されている．

このため，最小のコンピュータでも，モニタと称するものがあるとすれば，数万語に及ぶソフトウエアが付属することになる．大きなコンピュータでは，これが数十万語となり，大型のタイム・シェアリング・システム（time sharing system）では，百万語に達するものもある．

オペレーティング・システムの大型化の結果，コンピュータの実効的な性能は，金物そのものの性能からみれば，相対的には低下することになる．もっとも，利用者のプログラムを動かす場合以外は，すべて人手でやっていた時代では，その人手がかかる時間にはコンピュータが遊んでいたのであるから，それを勘定に入れて，性能の評価をしなければな

らないのであるが，普通そのような評価はされない．コンピュータの実稼動時間に対する，モニタの働く時間の占める割合が，運転諸掛り（overhead）として，システムを評価する1つの目安とされているが，これが数%から数十%に及ぶようになった．そのうえ記憶装置の相当部分がオペレーティング・システムに費やされ，利用者のための有効部分が狭まってきた．しかし，オペレーティング・システムは，その必要性のゆえに，コンピュータの応用分野が広がるにつれて，ますます大型化しているし，今後もその傾向は続くであろう．その結果，上記の問題点は，さらに深刻の度を深めることが予想され，いわゆるソフトウエア危機がコンピュータの発展のための重大な障害となってきた[1]．

ソフトウエア危機の重要な特徴は，ソフトウエア作製期間が長期化する傾向であろう．大型プログラムを作製する場合には，一般につぎのような問題がおこる．

（1） プログラムの虫

プログラム作製中の，不注意による誤りや，論理的な誤りは，ほとんど避けることができないうえに，その数はプログラムの大型化とともに，非直線的な割合で急増する．したがって，この誤りの修正，いわゆる虫とり（debugging）は，プログラムの大型化とともに，急速に困難な作業となっていく．

（2） プログラムの整合

上記の理由によって，1つのプログラムをあまり大型化することはできない．そこで，それを多数のプログラムに分割して，分業体制のもとに作製する必要がある．分業によってつくられた個々のプログラムが，1つのシステムにまとまって動くためには，プログラム間に一定の約束があって，それを正しく守るようにつくられなければならない．この約束は，一定の書式に従った文書によってなされるけれども，厳密な文書の記述方法がないために，理解の仕方にずれを生ずることが多い．その結果，各プログラムを合成してみると，どこかに誤りがあって，誤った動作をする．n個のプログラムがあれば，$N=n(n-1)/2$ 個のプログラムの対ができるから，最大N個の誤りの伝播通路がある．そこでこの誤りを発見するために必要な時間は，Nに比例するであろう．

（3） 保守上の問題

誤りのいくつかは，利用者の手もとにプログラムがわたって，使用されるうちに発見されるものもある．これらは，大型のオペレーティング・システムになると，その論理構成が非常に複雑であるため，工場内ですべての使い方について試験することは不可能なためである．これらの誤りは，初期の間に，急速に発見され修正されていくものでは

あるが，長い使用期間ののちに発見されるものも皆無ではない．通常は，このような場合の保守に，オペレーティング・システムの製作組織があたることは不可能であるので，特別な保守組織がつくられる．この場合にも，システムの大型化，複雑化が（1）の場合と同様な理由で，保守上の重大な障害となっている．

上記の問題点は，オペレーティング・システムの大型化によるものであるが，プログラム書きにアセンブラを使用していることが，その困難さを増幅している．アセンブラで書かれたプログラムの論理的な内容を読みとることは，そのプログラムの作製者以外では困難であるし，作製者自身でも，しばらくの後には読み取りにくくなる．アセンブリ言語は，人間の間の情報伝達の機能をほとんどもっていないと考えられる．その結果，上記（1），(2)，(3) のすべてにわたって，アセンブリ言語によるプログラムだけでは，その論理的内容を伝達できないので，通常は流れ図や，その他の説明的文書によって補足する．しかしこれらの付随的文書は，作製に際しては，その不断の更新なしにはプログラムを完成できない性質のものではないから，しばしば誤りを含んだまま放置される．虫とりや保守に際してこれらの誤りが，作業の大きなブレーキとなることはいうまでもない．

オペレーティング・システム作製の速度は，その大型化とともにしだいに減少する傾向がみられるのは，上記 (2) の理由にもとづくであろう．事実プログラマが1日に書くプログラムの命令数は，設計期間や虫とり期間全体にわたって平均すると，オペレーティング・システムが大きくなるにつれ，直線的に減少する様子をみせている（図 1.1）．これは，

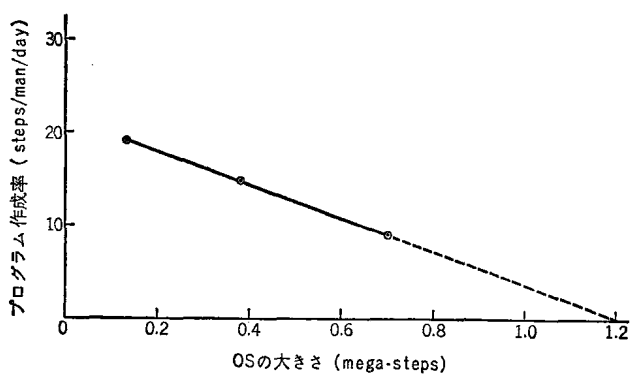

図 1.1　オペレーティング・システムの大きさに対するプログラマ1人1日のプログラム作製率
ただし，縦軸の単位は任意

オペレーティング・システムの大きさに上限を課するものである．システムの大型化にともなう記憶装置やコンピュータ運転時間の浪費は，記憶装置の大型化や，回路の高速化で補うこともできよう．しかし，図 1.1 に示される傾向は，ソフトウエア危機の，最も深刻な問題を示している．

このような問題を解決するか，少なくともやわらげるために，つぎの手段が考えられる．

1) オペレーティング・システムの構造にメスを入れること．最近の1つの方向は，オペレーティング・システムを構成する各プログラム・モジュール間の情報伝達を，すべてモニタ径由とすることである．これは，誤りの伝達をNの通路からn個に下げる効果があるが，モニタの働く時間を反比例的に増大させるので，別の難問題をひきおこす．

2) プログラム作製の際，誤りを生じないような手段を用いる．

3) 見かけ上，プログラムを小型にして，誤りの発生頻度を減らす．

2) の手段としては，オペレーティング・システム作製の自動化であろうが，これは現在では夢物語である．3) の手段としてはシステム・プログラム用言語が考えられる．

経験のある読者ならば FORTRAN や ALGOL などでプログラムをつくると，アセンブラでつくる場合にくらべて，10 分の1の短期間で完成することを知っているであろう．それは，これらの言語によるプログラムの1行は，だいたい機械語の 10 命令分に対応するからである．これは，プログラムを見かけ上，小さくするという効果をもっている．実際上，誤りの数も，同程度に減小している．そのうえ，これらの言語は，それだけで人間間の情報伝達に大きな力をもち，プログラムの論理を伝える表現力をもっている．だから，このような言語で記述したプログラムを，その内容を伝える文書とみなすことができる．このような経験は，当然，システム・プログラムの作製にも生かされるべきである．

実際に FORTRAN や ALGOL を使用して，たとえばコンパイラを書いた，いくつかの例がある．しかし，これらの言語は，システム書きのために工夫されたものではないから，あまりよい結果は得られない．すなわち，つくられたコンパイラは，アセンブラで書いたものにくらべて大きく，かつ，翻訳時間が長くなる．この点，最近に実用化された PL/I は，システム・プログラムを書くために，比較的都合のよい言語の形をしている．そこで，PL/I を用いてコンパイラを書く試みが行なわれるようになった[2]．けれども PL/I も，システム書き専用に設計されたものではないから，これで実用のシステムをつくろうとすればやはり種々の問題がおこる．そこで，システム・プログラムの大型化にともなう困難を克服する1つの重要な手段として，システム書きのための問題向き言語がど

うしても必要である.

さて，このようなシステム書き言語と，そのコンパイラの開発は，主として種々の言語のコンパイラ作製に使用することを目指して，実験的に進められているが，モニタ作製を意図する実験は，1,2 の例外を除いて，あまり見られない．それは，モニタの中で頻繁におこる割込み処理に必要とされる特別な効率の高さを，上記の言語が実現できるかどうか疑問視されるからであろう．このような問題はあるが，システム・プログラムの大部分がシステム用言語で書かれる時代は，目前にせまっているといえよう.

システム・プログラムのうちコンパイラは，言語処理という特別な性質のために，そのプログラム上の論理も，また他の部分と異なっている．さらに，他の部分からは，比較的に独立して，まとまった部分をなしているので，それだけで取り扱うことが容易である．そこで，コンパイラ作製を単純化するために，システム・プログラムの他の部分に対する手段とは異なった，コンパイラに適した方法を使用することが考えられる．さらにオペレーティング・システム全般に対しては夢物語である自動化の手段も，コンパイラに関しては若干事情が異なってくる．コンパイラ・コンパイラは，半自動化の方法の1つとみなしてよい．もっとも現在の段階では，コンパイラ・コンパイラの技術は，効率のよいコンパイラをつくり出すところに達していないから，実用化はまだ将来の問題である.

コンパイラ・コンパイラの研究が比較的盛んなのは，上記の実用的な観点や，それが単独に取り扱いやすいことのほかに，プログラム用言語開発のための道具として便利であることもある．コンピュータの応用分野が広がるにつれて，いろいろな問題向き言語が登場してきた．最初は FORTRAN，つぎに ALGOL や COBOL である．記号の処理のために，IPL-V，LISP，COMIT，SLIP，そして SNOBOL といった言語が工夫された．シミュレーションのために SIMSCRIPT，GPSS，GASP，DYNAMO，CSL，CQS，TRIM 等々がつくられた．応力解析のために FRAN，STRESS など，また，工作機械の数値制御テープの作製のための APT，最近ではグラフィックスの発達に従い，図形処理の言語がつくられているし，TSS の発達にともない，コンピュータと会話をしながらプログラムをつくることのできる言語等々，数えあげればきりがない．FORTRAN に統一されたかにみえた数値解析の分野でも ALGOL 60 が出現し，さらに FORTRAN，ALGOL，COBOL を包括し，リスト処理の機能を加えた PL/I がつくられ，ALGOL 60 の発展として，同じくリスト処理の機能を加えて，より現代的なコンピュータの発展に即応して，ALGOL 68，ALGOL N 等々が考えられている.

このような言語の処理プログラムをつくるには，大きな労力をともなうものであるから，その有効性，使いやすさ，翻訳上の問題点などについては，あらかじめ実験することが困難であるし，作製中または作製後に実験され，工合の悪いところを発見しても設計変更をするのは困難である．もし処理プログラムが簡単な言語で記述できれば，この難点が取り除かれ，作製されたコンパイラを用いて行なった実験の結果にもとづいて，言語自体の設計をやりなおすことが容易となる．したがって，今後ますます多様化していくはずの言語の開発にあたって，コンパイラ作製を簡易化する諸方法は，実験上便利な手段を提供する．現在までに開発されているコンパイラ・コンパイラは，ほとんどすべてこの目的のために使われている．

以上，コンパイラ・コンパイラの存在理由ともいうべきものについて説明してきたが，日本における，コンパイラ・コンパイラの開発は，ようやくはじまったばかりである．コンピュータ技術の先進国である米国では，多くのコンパイラ・コンパイラがつくられ，使用されているけれども，それは，ほとんど大学に集中していて，コンピュータ製造会社やソフトウエア会社では，CSC (Computer Science Corporation) の GENESYS[3] とか，COMPASS (Massachusetts Computer Associates) の TGS[4] とか，Texas Instruments の TMG[5] のほかには，あまり知られていない．それは，現在の技術が，上にのべた実験的手段以上に出ないためであろう．

引 用 文 献

1) 高橋秀俊，“ソフトウエア危機”，情報処理 10，No. 6，p. 373 (Nov. 1969).
2) 戸田巌，その他，“PL/I コンパイラ構成法の研究”，電気通信研究所研究実用化報告，18，No. 1，p. 115 (1969).
高橋延匡，その他，“HITAC 5020 TSS スーパバイザの PL/I による記述上の問題点”，昭和 44 年連合大会講演論文集 39，情報処理 (5)，3805，1969.
3) “THE GENESYS SYSTEM : A TECHNICAL INTRODUCTION,” The Computer Science Corporation, 1968.
4) J. Plaskow and S. Schuman, “The TRANGEN System on the M460 Computer,” AFCRL-66-516, 1966.
5) R. M. McClure, “TMG——a Syntax Directed Compiler,” Proc. ACM. 20th Natl. Conf., p. 262, 1965.

2. コンパイラ作製の簡易化

コンパイラ作製を簡易化する，いろいろな方法がある．ここでは，すでに試みられているそれらの方法について，簡単に解説してみよう．

2.1 コンパイラ向き記述言語

すでにのべたようにアセンブリ語を用いるよりは，コンパイラの翻訳手続きを簡明に記述しうる高級な記述言語を使用するならば，コンパイラ作製の労力を数分の1に減らすことができよう．

現在までに，FORTRAN コンパイラを FORTRAN 語を用いて記述したり，ALGOL コンパイラを ALGOL 語を用いて記述した若干の例がある．コンパイラの手順は，数値計算の場合とちがって，複雑な数式処理はほとんど不必要であるけれども，データの量を節約するために，ビットやバイトを取り扱う必要があるし，条件の調べ方が複雑である．たとえば ALGOL 語で ALGOL コンパイラを書いたある実験[1] では，代入文の使用頻度は全体のうち 32% を占めて，比較的高いけれども，それらの右辺にそれぞれ2個の乗算と加算を含む式やそれ以上に複雑な式はあらわれていないし，項はたかだか3個である．代入文のほぼ 80% は，

$$X:=Y\pm Z$$

か，またはそれ以下の単純な式である．ただしXは変数，Y, Z は変数か常数である．条件文の数は全体の 19% であるが，表現型式の変化が多い．条件節の中で，論理和や論理積をとる場合が多いし，数個の2進桁を同時に調べる必要がある．また字の列を取り扱う必要もあるし，型やその他の性質の異なる要素を1行に含む行列も取り扱う必要がある．

ALGOL や FORTRAN は，上記のような取り扱い方に適していないから，これらの

言語でコンパイラを書くとすると，まわりくどい方法を用いなければならない．その結果，表現が複雑になるし，つくられたコンパイラの効率はあまりよくない．†

それゆえ，コンパイラ記述に適当な新言語を考える必要がある．その1つとしては，PL/I を利用することである．PL/I は，ビットや字の列を取り扱ったり，複雑なデータ構造を表現する手段をもっているので，コンパイラを書くに適している．実際に，コンパイラを中心としたシステム・プログラム書きのために，PL/I のサブ・セットの開発が進められている．††

しかし PL/I は，システム・プログラムやコンパイラのための専用の言語として設計されたものではない．数値計算や事務用の応用も含んだ，汎用の言語である．それゆえ，その表現形式はシステム・プログラムやコンパイラの狭い側からながめると，一般的すぎて，直観性を欠くきらいがある．

コンパイラ記述言語を考える場合，もう1つ重要な問題がある．このような言語はアセンブリ語と異なり，印刷物のうえで2次元的配列も考慮に入れうるし，なによりも1行1行の顔つきが異なって，代入文や条件文が使用されるため，書きやすく，読みやすいけれども，それでも流れ図のもつ直観性には及ばない．それゆえ，保守に必要な記録文書としては，どうしても流れ図が含まれなければならない．そこで，言語の表現形式が，流れ図に含まれる図式と1対1対応のつくものであれば，流れ図とプログラムの照合は大変容易となって，製作上はもちろん，保守上，大きな便宜を与えるであろう．

次節の後半で，この種の言語によるコンパイラ生産の過程を，形式的にあらわしてみよう．

2.2　コンパイラの変換

通常，ある型のコンピュータには，いろいろな言語のコンパイラがつくられているが，新しく開発されたコンピュータに，同じ言語のコンパイラをつくらなければならない，という場合が多い．このような場合に，古い機械上に働くコンパイラを使用して，新しい機械上に働くコンパイラをつくり出すことができれば大変都合がよい．このための方法をつ

† ここで効率とは，コンパイラの，機械語で勘定した語数とそれの翻訳の速度による，あいまいな評価概念である．

†† ここで，サブ・セットというのは，本来の PL/I から，いろいろな表現上の機能を取り去ったものをあらわす．実際に試みられているものは，サブ・セットに，PL/I に含まれない表現が加わったものもある．文献 1. の 1 参照．