博士論文

プログラム言語の日本語化と国際化に関する研究

2002 年 3 月

今城哲二

本論文は奈良先端科学技術大学院大学情報科学研究科に博士（工学）授与の要件として提出した博士論文である．

今城哲二

審査委員：　植村俊亮　教授
渡邉勝正　教授
松本健一　教授
吉川正俊　助教授

プログラム言語の日本語化と国際化に関する研究*

今城哲二

内容梗概

約 20 年前からコンピュータでの漢字利用が普及し，標準プログラム言語で日本語データ処理が可能となった．日本語処理は，国際化（internationalization）と一般化されて国際規格化され，いくつかの言語では識別名にも漢字などマルチオクテット文字が使用可能となっている．予約語まで日本語にした本格的な日本語プログラム言語も，分かち書きのレベルで実用化されているが，非分かち書きレベルはまだ研究段階であり，複数のグループで研究が進行している．

本論文では，プログラム言語の日本語化と国際化について，本文で筆者が携わった次の三つの研究内容と成果を論じ，付録で過去 40 年間のこの分野の研究と商用化状況に関する文献を網羅的に紹介する．

(1) COBOL 日本語機能の言語仕様を開発し，これをマルチオクテット文字機能さらには国際化機能として一般化した．その成果を国際規格として提案し，採用されることが決定している．標準化にあたっての課題は，日本という地域性を排除して，国際的に通用する技術とすること，および日本語機能を用いた既存資産の移行性を確保することの二つだった．次の方針を策定実行して，将来にわたって使用可能な汎用的な国際化機能の言語仕様を制定することに成功した．

(a) 対象とする文字をアルファベット 26 文字以外の世界各国の文字に拡張するとともに，多くの符号系に対応可能な普遍的な枠組を設定する．

(b) 既存の文字型に加えて，漢字などの文字型を新設する．それらの型を処理する文は同等とし，同じ業務プログラムを各国ごとに適用するときに

＊ 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博文論文，NAIST-IS-DT9761201，2002 年 3 月 7 日

は，データ定義だけを修正しロジックの変更は不要とする仕様とする．

(c) 国内で普及している日本語処理機能を，国際化機能の核として採用する．

(2) 10 年ほど前に，第 4 世代言語（4GL）はメインフレームの業務アプリケーションを効率よく構築する手段として注目を浴びていた．我々が開発した EAGLE/4GL は，その構文に日本語（分かち書き）を採用したことにより，プログラム作成が楽になるとともに保守性を大幅に向上させることができ，万を超えるユーザに利用された．また，メインフレームのオンライン制御プログラム環境（OLTP 環境）と対話環境（TSS 環境）の両方で動作可能なアプリケーションを作るため, 環境独立なオンライン業務プログラム処理方式を採用した．現在，第 4 世代言語はアプリケーション開発の主流ではないが，環境独立処理方式は現代のオープン環境にも有効に適用できる．

(3) 分かち書きをしないより日本語に近い日本語プログラム言語 “ まほろば ” の言語仕様を設計した. この言語の設計方針は次の四つとし, 記述実験を行い, いずれの項目もほぼ達成していることを実証するとともに，今後の課題を指摘できた．

(a) 日本語として違和感のない仕様とする．

(b) 標準プログラム言語で記述できる “ データ構造とアルゴリズム ” を記述可能とする．

(c) 言語普及を目指すために，市場が大きい事務処理分野で適用可能とする．

(d) プログラム品質に悪影響のある GOTO 文とポインタ変数は支援しない．

これらの研究は，プログラム言語の日本語化の主要な三つの発展段階に対応し，それぞれの段階で代表的なものの一つとして，各分野に大きな影響を与えてきた．さらに今後のこの分野の研究に寄与することが期待される．

キーワード プログラム言語，日本語機能，マルチオクテット文字機能，国際化，COBOL，第 4 世代言語，日本語プログラム言語，環境独立処理方式，オンライン業務プログラム

Studies on Japanese-based Programming Language and its Internationalization *

Tetsuji Imajo

Abstract

20 years ago, programming languages have acquired an ability to handle Japanese. Generalizing it to internationalization (i18n), many of i18n facilities of each programming language were accepted as ISO standards. Some programming languages allowed users to use user-defined words in such multi-octet characters as kanji. Some new Japanese-based programming languages have also been developed and used. In all those languages, keywords and grammar are based on Japanese, but words have to be separated by spaces. Non-separated (No wakachigaki) Japanese-based programming languages have been researched by vendors and universities.

In this paper, contents and results of author's three following researches on Japanese-based programming languages and its i18n are reported, and an appendix includes survey of papers on their fields

.

(1) The author proposed the internationalization facility of COBOL by generalizing the Japanese facility. The facility was decided to be accepted as an International Standard. There were two goals need to be achieved for this standardization: removing Japanese locality in order to obtain world wide approval, and ensuring compatibility with the Japanese facility used in existing user programs. The following principles allowed us to make the specifications for internationalization that can be generally used for long.

(a) The facility should accept any coded character sets used in the world.
(b) The operations for the newly introduced international data type should be identical to those for alphanumeric.
(c) Features that already have been used widely should be adopted as the core of the new COBOL internationalization facility.

*Doctor's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9761201, March 7, 2002.

(2) About 10 years ago, the 4th generation languages (4GL) were used as an efficient way to develop batch and online application programs. Our 4GL, EAGLE/4GL, supported the Japanese-based syntax with wakachigaki version, thus the 4GL programs are written and read easily in Japan. EAGLE/4GL supported environment-independent online application program pattern to be used both OLTP environments and interactive environments on mainframes. Now 4GL is not a main stream of application development, but the idea of environment-independent pattern is still important and efficient on today's open environments.

(3) Language specification of “Mahoroba”, Japanese-based programming language with non-wakachigaki version, was developed by the author. The design goal of this language was set to the following four points, and all of them were confirmed to be achieved by coding tests.

(a) Source code in “Mahoroba” must be recognized as natural Japanese sentence.
(b) Data structures and algorithms in common programming languages should be able to be described with “Mahoroba”.
(c) For wide adoption, practical business application must be able to be written by the language.
(d) Avoid supporting GO TO statement and the pointer variable which have bad influences for software quality.

Each of these three researches describes the typical topic in the stages of development of Japanese-based programming languages. Thus, these researches should contribute to the future study in this field,

Key words: programming language, Japanese facility, multi-octet character handling facility, internationalization, i18n, COBOL, the 4th generation language, Japanese programming language, environment-independent program pattern, online application program

目次

図目次

表目次

第１章　序論

プログラム言語の中に日本語を取り入れる研究と開発は 1960 年代から始まっており，1980 年代に日本語処理システムが実用段階になるに至って本格化した．最初は漢字データ処理が中心テーマであったが，ソースプログラムでも日本語を利用したいとのニーズから, 現在では COBOL, SQL, Prolog, ISLISP, C++の標準言語で利用者定義の識別名として日本語（漢字）の指定が可能となっている[今城 99c]．これらの日本語処理は，国際化（internationalization）と一般化されて国際規格化され, いくつかの言語では識別名にも漢字などマルチオクテット文字が使用可能となっている．

1980 年代後半には，さらに日本語化が進み，予約語も含めて日本語にした本格的な日本語プログラム言語が実用化された．ただ，日本語といっても分かち書きを前提としていた[機シ振協 88]，[玉井 90]，[今城 99a]．分かち書きがないと, コンパイラの字句解析処理で語を認識するのが格段に難しくなるからである．

その後日本語プログラム言語の研究・開発は一時下火になったが，近頃はいくつかのチームにより本格的な日本語プログラム言語の研究が進んでいる[プロシン 99]，[宮脇 94]，[宮脇 99]，[中鉢 97]，[鈴木弘 99]，[加藤木 99]，[畠山 2000]，[兼宗 2001b]．この中で，分かち書きをしないものは宮脇と中鉢の研究であるが，中鉢の研究はまだ完成していない．宮脇の日本語 C++は C++を忠実に日本語化したものであり, 専用エディタを用いてコンパイラの字句解析での語の認識の難しさを回避している．プログラマは日本語 C++専用エディタでソースプログラムを編集するが，そのとき語の切れ目を指定する．専用エディタはその切れ目情報をソースファイルに反映し，日本語 C++コンパイラは字句解析時にその情報を参照しながら語の切り出しを行う．

本論文では，プログラム言語の日本語化と国際化について，以上の三つの研究分野に対応し，次の三つの課題について筆者の研究をまとめたものである．

(1)　COBOL 国際化機能の言語仕様の開発と標準化

COBOL の日本語化機能を JIS にする計画があったが，日本独自の標準が非

関税障壁とみなされ貿易摩擦の悪化の要因になることを避けるため，日本語機能をマルチオクテット文字機能さらには国際化機能として一般化し，日本からその仕様を国際規格として提案することにした．標準化にあたっての課題は，日本という地域性を排除して，国際的に通用する技術とすること，および日本語機能を用いた既存資産の移行性を確保することの二つであった．次の方針を策定し，それにそって将来にわたって使用可能な汎用的な国際化機能の言語仕様を制定することが課題であった．

(a) 対象とする文字をアルファベット 26 文字以外の世界各国の文字に拡張する．

(b) 既存の文字型に加えて，漢字などの文字型を新設する．それらの型を処理する文は同等とし，同じ業務プログラムを各国ごとに適用するときには，データ定義だけを修正しロジック変更は不要になるような仕様とする．

(c) 国内で普及している日本語処理機能を，国際化機能の核として採用する．

(2) 第 4 世代言語における日本語プログラミング

10 年ほど前に，第 4 世代言語（4GL）はメインフレームの業務アプリケーションを効率よくを構築する手段として注目を浴びていた．我々が開発した EAGLE/4GL では，生産性向上のためいくつかの特長も持たせた．その中の代表的なものが，日本語構文（分かち書き）の採用と業務のパターン化である．これらにより，プログラム作成が楽になるとともに生産性と保守性を大幅に向上させることを目標とした．

(3) より本格的な日本語プログラム言語の研究

分かち書きをしない，より日本語に近い日本語プログラム言語 “ まほろば ” の言語仕様を設計した．これは宮脇の日本語 C++とは異なり新規の言語として設計した. 分かち書きをしない言語であるが, 専用エディタは不要とし, “ 通常のテキストファイルのソースプログラムをコンパイラが入力して, 字句解析できること ”を要件とした．そのためには語の切り出しのための新たな構文解析手法が必要であった．

この言語の設計方針は次の四つであり, これらが満たされているかどうかの記述評価も行う必要があった．

(a)　日本語として違和感のない仕様とする．

(b)　標準プログラム言語で記述できる“ データ構造とアルゴリズム ”を記述可能とする．

(c)　言語普及を目指すために，市場が大きい事務処理分野で適用可能とする．

(d)　プログラム品質に悪影響のあるGOTO文とポインタ変数は支援しない．

本論文の構成は次のとおりである．**2.**では，標準プログラム言語の国際化機能について，COBOLの標準化を中心に述べる． **3.**では，分かち書きの日本語プログラム言語 EAGLE/4GL の開発思想，仕様を説明し，そこでの環境独立オンライン業務プログラム方式について論じる． **4.**では，分かち書きをしない日本語プログラム言語“ まほろば ”の言語仕様を定義し，その評価について述べる．**5.**は本論文のまとめである．**付録 A** で,過去 40 年間のプログラム言語の国際化と日本語化に関する研究と商用化状況に関する文献を網羅的に紹介する．**付録 B** で，まほろばの記述実験用に作成した“ まほろば ”のサブセットである“ まほろば 0 ”の文法と処理系を説明し，そのソースリストを示す．

なお，筆者の役割はそれぞれ以下のとおりである．

(1)　COBOL：情報処理学会日本語機能専門委員会の委員として標準化の方向付けに参加．情報処理学会 SC22/COBOL WG 主査として国際化機能をCODASYL および ISO に提案．CODASYL，ISO/IEC/JTC1 および米国の COBOL 標準化委員会の委員として審議．

(2)　EAGLE/4GL：当該製品の日立製作所の開発プロジェクト責任者．

(3)　まほろば：奈良先端科学技術大学院大学での研究として当該言語を 1 人で開発．

第2章　COBOL国際化機能の開発とその国際標準化

2.1　はじめに

情報処理分野の標準化の過程では，多くの新しい概念や仕様が提案されて，研究や学問の進歩にさまざまの影響を与えているが，その内容が学会の論文誌に報告されることは，ほとんどなかった．本章では，COBOL日本語機能（のちに，国際化機能）を国際規格として提案し，その制定をはかる過程を，プログラム言語システムの開発としてとらえて，言語仕様の決定に関する技術課題を中心に報告し，仕様の有効性を示す．

文中では，ISO/IEC による国際規格を制定するまでの作業を国際標準化あるいは単に標準化という．

各国・各地域の文字，文字の照合順序，通貨記号，数字の表現，日付・時刻の表現など文化的な要素を計算機で処理可能にする一般的な枠組みを，国際化機能という．

1 文字の表現に 8 ビットの倍数ビットを使う符号系をマルチ（多）オクテット符号系あるいは多バイト符号系などという．情報技術の標準化では，1 バイトが 8 ビットであることを明示するために，オクテットという用語をよく用いる．

COBOL 日本語機能の国際標準化にあたっては，地域性の排除と，旧来の日本語機能による資産の継承という，一見矛盾する二つの要請があった．これまでによく使われてきた日本語機能を核としながら，世界の符号系を対象とする普遍的な枠組みを設定する言語仕様の設計と，その国際標準化に成功した．

この章の構成は次のとおりである．**2.2** では，国際標準化の手順，COBOL および他プログラム言語の国際化機能の標準化経緯，関連研究をまとめる．とくに本格的に国際化機能を実現しているＣの国際化機能の考え方について論じる．**2.3** では COBOL 国際化機能の仕様策定について，先の二つの課題とその対策を述べる．**2.4** は，国際規格として採用されることになった国際化機能の内容である．**2.5** では国際化機能の適用プログラム例を示し，国際化機能の有効性を論じる．**2.6** では今回提案した国際化機能では未解決の今後の課題を明らかにする．**2.7** では本章のまとめを述べる．

2.2　標準化の経緯，条件および関連研究

2.2.1　標準化の経緯と前提条件

1980年前後に漢字プリンタや漢字端末が普及し，これにともないCOBOL, Fortranなどのプログラム言語で日本語機能が導入されたが，言語仕様はベンダごとに異なり互換性がなかった．とくに，国産ベンダと海外ベンダの仕様差が大きく，利用者から仕様統一の要求が多く出された．

1986年には，日本語FortranのJIS原案が完成したが[日本語FORT]，国際規格と異なるJISを制定するのは海外から非関税輸入障壁とみなされるとの懸念から，制定直前にJISにすることは取りやめになった．これを契機に，日本語処理の標準化に関する産官の方向が，「日本語（漢字）機能を一般化した“マルチオクテット文字処理機能”を国際規格に追加し，その後でそれをJISにする」と変わっていった．

1986年以降は，この方針に添って，情報技術の国際規格の国内審議機関である情報処理学会情報規格調査会（IPSJ/ITSCJ）が，各分野でマルチオクテット文字処理機能の国際規格化を推進した．プログラム言語については，国内のITSCJ/SC22委員会と日本語機能専門委員会（筆者はこれらの委員会の委員）が1987年のISO/IEC/JTC1/SC22会議で「各プログラム言語の規格にマルチオクテット文字処理機能は必須とする」という提案をして，全会一致で承認された[日本語委86][中田87]．これを受けて，JTC1/SC22の各プログラム言語の委員会でマルチオクテット文字処理機能の検討が本格化し，わが国は主導的に作業した．

1990年以降，国際規格及びJISが順次制定されている[今城99c]，[今城99d]．この段階では，各国・各地域の文字だけでなくて，文字の照合順序，通貨記号，数字の表現，日付・時刻の表現などの文化的な相違も標準化において考慮することが重要であるとの認識が進み，マルチオクテット文字処理機能はより一般化した「国際化機能」と総称されるようになった．

国際規格を制定するまでの標準化の作業には，慎重な技術的配慮と，手順が必要である．COBOLに新しい言語仕様を採用する手順は他の言語とは異なる．まずアメリカのCODASYL COBOL委員会が仕様を決め，それからアメリカ規格，国際規格，JISなどの標準化された言語仕様になる．

マルチオクテット文字処理機能の標準化のため，国内のCOBOL標準化委員会のITSCJ/SC22/COBOL WG（筆者が主査）は1988年以来CODASYL COBOL委員会に参加し，仕様を作成し，1992年に承認された[CODASYL94].

国際規格の原案作成は，ISO/IEC/JTC1から，アメリカのCOBOL標準化委員会であるNCITS/J4に委託された．ITSCJ/SC22/COBOL WGは，1992年にNCITS/J4のメンバになった．2002年に制定が予定されている次期COBOL国際規格では，CODASYLのマルチオクテット文字処理機能を基本として，これを国際化機能としてより一般化し，ISO/IEC 10646（ユニコード）[JISUCS95]の支援とC言語及び POSIXの地域固有機能（locale）に対応する機能などの追加・改善を行い，すでに国際化機能についての技術的な審議を終了している[ISOCOBOL2001].

COBOL日本語機能の規格案の開発と提案においては，すでに述べたように課題が二つあり，それが大前提として言語仕様開発の制約条件となった.

(1) 国際規格は，世界各国にとって公平な仕様にしない限り各国の賛同を獲得できない.

(2) 利用者のプログラム資産はほとんどCOBOLだといわれており，この資産との共存と移行性確保が，COBOL規格の開発における重要な要件である．標準でない日本語機能を利用したプログラム資産は年年膨大な量が開発されており，これから標準的な言語仕様への移行工数が最小で済むように仕様を決められるかどうかが規格としての成功の鍵をにぎっていた.

2.2.2 関連研究（他プログラム言語の標準化動向）

COBOL以外のプログラム言語のマルチオクテット文字処理機能は，各ベンダの実装が先行し，その言語仕様が日本国内や国際的なコンソーシアムの中で実質的な標準（ディファクト標準）として議論され，最終的には国際化機能として国際規格（デジュール標準）仕様として採用された．国際化に関する標準化のガイドラインとしては，JTC1の技術報告[TR国際化2000]と[TR言語規格2000]がある．[今城99a]は，国内の学会の全国大会や研究会で発表された多くの文献を紹介している．各プログラム言語の国際化機能の技術的内容は，**表2.1**に示す国際規格あるいはJISが基本文献である．POSIXとC言語の国際化機能に

表2.1　国際化機能を含むプログラム言語と規格番号

言語	ISO/IEC	JIS
C	9899:1990 Amendment1:1995	X 3010-1993 X3010-1996(追補１)
Fortran	1539:1991	X 3001-1996
SQL SQL2	―― 9075:1992	X 3050-1990 X 3005-1995
POSIX	9945-1:1990 9945-2:1993	X 3030-1994 作成中
Ada第2版	8652:1995	作成中
Prolog	13211-1:1995	作成中
ISLISP	13816:1997	X 3012-1998
C++	14482:1998	作成中
COBOL	2002年制定見込み (CODASYLは1992年)	作成中

ついては[清兼98]に詳しい.

POSIXとCの国際化機能の特長は，ロケールという概念と機能が採用されていることである．これは，システムソフトウェアインタフェースの国際標準化を目的としJTC1中に設立したTSG-1の最終報告[JTC1TSG91]に記述されており，日本がそこで提案した国際化モデルである．このモデルは，「国際化・現地語化（localization）モデル」といい，CとPOSIX，さらにC++のロケール仕様に反映された．**2.4.4** に述べるCOBOLのロケール仕様も，同一システム上でのC及びPOSIXと共存するときの整合性を重視し，それらの仕様に極力一致するように努めている．

標準プログラム言語の中で国際化機能について最も本格的に取り組んでいるのは，COBOL，CおよびC++である．C++の国際化機能のほとんどはCを踏襲している．ここでは， Cの国際化機能の基本部分を紹介し，問題点をあげる．

Cの標準入出力，ファイル，プログラム上の文字に関する仕様は，1バイトが前提になっていた．マルチオクテット文字を扱うときは，多バイト機能，すなわち国際化機能を使用する．多バイト文字には，1バイト文字も含むと定義されている．多バイト機能では，標準入出力やファイル上で，文字列が多バイト文字列として存在すると考える．多バイト文字は，1バイトであろうと2バイト以上の文字であろうと入力時に同じバイト数のワイド文字に変換し，プログラ

ム上ではワイド文字だけを扱い，出力時には多バイト文字に変換して標準出力やファイルに出力する．こうすることにより，シングルオクテット文字とマルチオクテット文字の間のシフトコードをプログラム上で意識不要とするだけでなく,文字の長さを気にせずにプログラムが書けるようになる.ワイド文字(列)を扱う関数は，1バイト文字（列）用に対応し異なる名前で一式（約60個）用意している．この概念図を図2.1に示す.

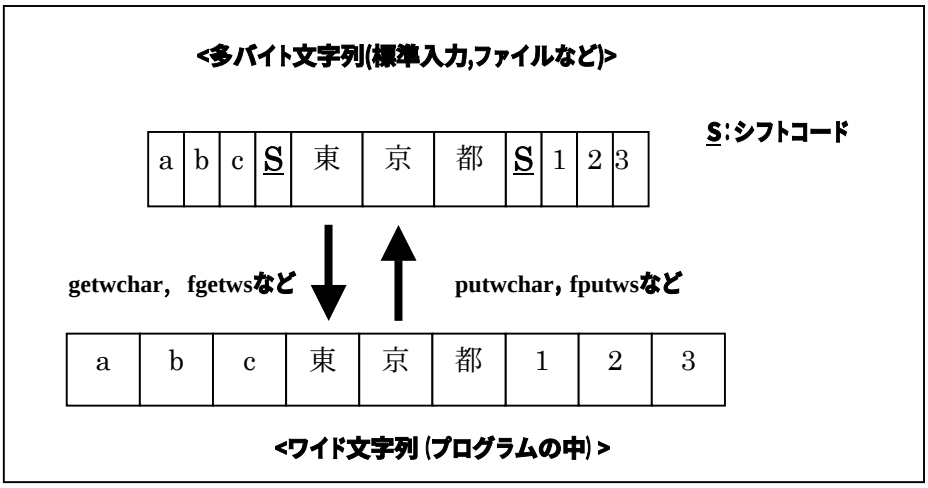


図2.1　C言語の多バイト文字とワイド文字

このような方式を採用したのは，C ではテキストファイルの文字列を先頭から１文字ずつ文字を調べながら処理する１バイト文字（列）用のアルゴリズムが普及しており，1 バイトを含む多バイト文字（列）用にも同じアルゴリズムを適用したいというニーズが高いことによる．このため，ワイド文字というデータ型を導入し，文字のバイト数の違いを意識不要とした．しかし，関数の型および各引数の型は一つしか取れないという C 言語の文法上の制約から，機能は１バイト文字（列）用関数と対応するが名前が異なるワイド文字（列）用の関数を用意することになった．この結果，1 バイト文字用のプログラムを多バイト用に修正するとき,文字型や定数の内容を変更するだけでなく，(ロジックの流れの修正は不要だが）ロジック中で文字（列）を参照する関数名をすべて変更する必要がある.

後の節で詳述するように，COBOL の文字処理は文と組込み関数で行ってお

り，それらのオペランドのデータ型は１種類に限定しなくてもよいという文法特性があるため，同じ構文で１バイト文字列も多バイト文字列も記述可能であり，ロジック部分の修正は不要である．

2.3　国際標準化の課題と仕様策定方針

2.3.1　課題と方針の概要

2.2.1で述べたように，COBOL日本語機能をCODASYL仕様，続いて国際規格のための仕様として提案するにあたっては，二つの主要課題があった．

⑴ 日本固有の地域性を極力排除した公平な汎用仕様にしなければならない．

⑵ 既存の日本語機能を使用した膨大な資産からの移行性を確保する必要がある．

この矛盾する課題に対して決めた仕様策定方針を次項以降に詳述する．

2.3.2　世界各国・地域の文字

国際標準化の最大問題の一つは，それが日本だけでなく世界各国にとって有益であることを納得してもらうことである．このため，対象となる文字の範囲を日本語の文字だけでなく，中国の漢字，韓国のハングル文字，タイ文字などアジアのマルチオクテット文字や，ドイツ，ロシア，北欧，東欧諸国など欧州各国で用いるアルファベット26文字以外の文字（これらは8ビットすなわちシングルオクテットで収まる）も含めた広い範囲に拡張する方針とした（**図2.2**）．マルチオクテット文字の定義は，シングルオクテット文字と同じまたはその倍数とし，これらのマルチオクテット文字を標準プログラム言語で使用できるようにすることは国際規格としての必要条件であると強く提唱し，各国の賛同を得るように努めた．

日本では，JIS漢字符号系が制定されている．ベンダの漢字符号系は,これに準拠しているものやいないものなど，さまざまであった（「準拠」ということば自身が標準化の立場からはあいまいであることも，指摘しておきたい）．中国や韓国などアジア各国のマルチオクテット文字符号系は，各国の規格としてそれぞれ制定されており，欧州各国に特有のシングルオクテット文字も各国規格お

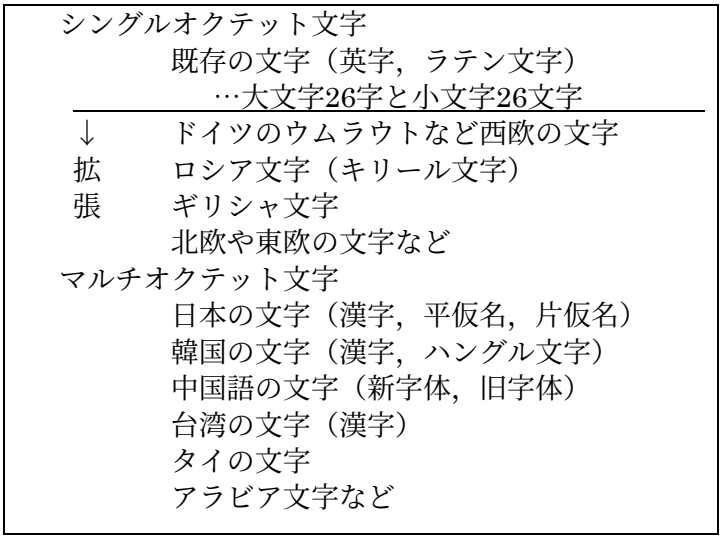


図2.2　COBOLで使用可能とすべき各国・地域の文字

2.3.3　普遍的な符号系の枠組み

よび国際規格(8589-1など)として制定されている．1980年代後半は存在していなかったユニコードは1993年にISO/IEC 10646として国際規格となった．これらの規格は不動のものではなく，たびたび改訂される．今後も各種の符号系が実用化され, その中から新たな規格が制定されていく. 個々の符号系に偏らず, 長い将来にわたって適用可能な普遍的な符号系の枠組を設定することが重要である．

他のプログラム言語規格では，符号系については規格書では言及せずに，既存のアルファベット26文字以外の文字を一般的に拡張文字，マルチオクテット文字，あるいはマルチバイト文字とよび，どのような文字を支援するかは，コンパイラ作成者依存事項としている．COBOLでは，普遍的な符号系の枠組みを設定するとともに，たとえばEBCDICファイルをASCIIの世界でも処理可能とするなど事務処理に頻繁にあらわれる問題も解決できるように，個別の符号系を意識した指定もできるようにしなければならない．

2.3.4 既存資産との共存

COBOLでは，膨大なシングルオクテット文字の資産が存在する．これらとマルチオクテット文字の資産が上手に共存する必要がある．そのためには，既存のシングルオクテット系符号系とマルチオクテット系符号系が共存することを前提に仕様を考える．これらの符号系共存時には，ファイルや通信上に符号系切替用のシフトコードが存在する符号化法と，存在しない符号化法の二つがありうる．シフトコードの存在を意識したプログラムのロジックは難しく，シフトコードの値も符号化法によって異なるので互換性のあるプログラムを作成するのは難しい．これを避けるため，ファイルなどプログラムの外ではシフトコードは存在するが，プログラム上ではその存在を意識しないような方式とすることを方針とした．

2.3.5 文字データ型

COBOLの既存の文字型として英数字項目がある．これは，ASCIIやEBCDICなど1バイト系符号系を暗黙に前提としている．これに対して，マルチオクテット文字を収容する文字型として国別文字項目（national character item）を新設し，英数字項目を扱う文と各国文字項目を扱う文の書き方は同じとする方針をとった．COBOLでは転記や比較をはじめとした多くの文で英数字項目を扱えるが，そのすべての文で同等に各国文字項目を支援できるようにするのはコンパイラの負担が大変であったため，実装では恣意的に制限されている面があった．これでは，英数字項目だけで十分なイギリスやアメリカと比べ，各国文字項目を使用する他の国・地域は不公平となるので，英数字項目を扱う機能と各国文字項目を扱う機能を同等にすることを方針とした．

国別文字という用語にした理由は，アルファベット以外のヨーロッパ各国のシングルオクテット文字も対象としていることを常に意識するためであり，言語仕様ではマルチオクテット文字やマルチバイト文字という用語は意識して使用しないことにした．

2.3.6　各国・地域に適用可能なプログラムの記述

たとえば英語用に開発したプログラムを,日本語用など他の国・地域用に容易に変更可能にしたいという要請がある．このためには，開発当初から各国用に変更すべき箇所を局所化し，それ以外は共通にすればよい．このようなプログラム作成方式を“国際化プログラム作成法”という．COBOLの国際化プログラム作成法では，局所化すべき変更箇所はファイルやデータなど国・地域に関する項目と定数定義部分だけにし，手続き部の中の文や関数のロジック変更がないようにすることを方針とした．このためにも，英数字項目に関する文や関数の構文が国別文字項目に関するそれらと全く同じであることが望ましい．これはC，C++，Fortranでは実現できていない．なお，この国際化プログラムの重要性は，CODASYL COBOLへの提案時には認識が低かったが，POSIXやCの標準化動向からの影響により，国際規格への提案時には重要と認識するように変わった．

2.3.7　在来型日本語機能の活用

標準化に時間がかかっている間に，膨大な日本語処理プログラムの資産が増大・蓄積されている．国際規格を作っても，実際に使われている仕様と異なると移行費用・期間がかかるので，その仕様は普及しない．これでは規格を作る意味がないので，各ベンダの既存日本語機能の中で使用頻度の高いものを共通仕様として選定し，それらを国際規格の核となる構文として採用することを方針とした．

各ベンダの既存の仕様には，プリンタの活字の大きさやフォント指定など，地域性・時代制約性・ハードウェア依存性の高い仕様が含まれていた．これらは，汎用性がないので，国際規格案には含めないこととした．ただ，PICTURE句のN指定は多用されているので，既存財産維持のため規格仕様として採用する必要があった．これは，日本語(Nihongo)のNではなく各国（National）文字指定のNと解釈することにした．具体的に採用したのは，**図2.3**の項目であり，大部分の日本語処理プログラムはこれらだけを使用している．若干書き方が違う実装があるが，簡単なコンバータで機械的に変更可能である．

- 各国文字項目を指定するPICTURE句のN.
- 各国文字定数の書き方　N”…”.
- 各国文字項目の転記と比較.
- 各国文字項目と各国文字定数の先頭と最後にシフトコードは含まない.
- プリンタなどにWRITE文で印字時に，必要ならば各国文字項目の前後にシフトコードを自動挿入する.

図2.3　おもな在来型日本語機能

2.4.　国際化機能の国際規格

2.4.1　文字集合と文字の種類

COBOL国際化機能の国際規格は，他のプログラム言語の規格書に比べて，“文字集合と文字の種類”についてはるかに厳密に規定している．また，プログラムの記述と動作が，シフトコードが存在する環境（主にメインフレーム）と存在しない環境（主にUNIXやPC）とで同一になるように，仕様を工夫している.

文字を何ビットの大きさにするかはコンパイラ作成者定義事項なので，英数字のための符号系として英数字を含んでいるJIS漢字符号系（14ビット）あるいは10646のUCS2（16ビット）やUCS4（32ビット）を実装することは理論上は可能である．こうすると，漢字なども英数字と同じ大きさの文字として取り扱うことが可能となり，特別な仕様を追加せずにマルチオクテット文字処理の問題はほとんど解決する．しかし，現実には1バイト符号系の膨大な利用者の財産が保存・活用されているので，それからの移行性がなくなる実装を国際標準COBOLで推奨することはできない．このため，文字集合として次の二つを用意することにした.

(1) 英数字文字集合： 英数字文字集合という用語は，国別文字集合とともに新しい用語であるが，以前の規格書の計算機文字集合と同じで，COBOLのソースプログラムや実行時に使用されていた文字集合である.現在の実装としては，ASCIIやEBCDICなどの1バイト符号系（文字の大きさは7ビットまたは8ビット）を暗黙に想定している.

⑵ 国別文字集合： マルチオクテット文字の文字集合をCOBOLで扱うために国別文字集合という概念を導入した. 具体的な符号系として, 10646, 日本, 韓国，中国などの国の規格として制定されている符号系，各ベンダ固有の符号系，シフトJISやEUCなど，いずれを実装しても共通仕様になるように考えた概念である．欧州各国にある，アルファベット26文字以外のシングルオクテット文字符号系にも対応できるように，マルチオクテットやマルチバイトという用語は使わずに国別文字という用語を使うことにした.

2.4.1.2 二つの文字集合の関係

英数字文字集合と国別文字集合との関係を，明確に規定した.

⑴ 二つが一個の統合（integrated）文字集合を構成.

例：英数字文字集合は10646のUCS2のサブセットで，国別文字集合は10646のUCS2全体．理想的な形態だが，当分の間現実的ではない,

⑵ 二つが合成（composite）文字集合を構成.

例：バイト中のビットパターンで1バイト文字（英数字文字集合）か2バイト文字（国別文字集合）かを区別し，シフトコードを用いずに，符号系間の切替えをする. シフトJISやEUCのように, UNIXやPCで普及している形態である.

⑶ 二つが別個な文字集合で，それらが混在（mixed）.

例：JIS漢字符号系とASCII符号系が混在し，それらはシフトコードによって区切られる．メインフレームで普及している形態である.

2.4.1.3 文字の大きさ

文字の大きさ（ビット数とバイト数）は次のように規定した.

⑴ 英数字文字集合のそれぞれの文字は同じバイト数で表現する.

⑵ 国別文字集合のそれぞれの文字は同じバイト数で表現する.

⑶ 国別文字集合の文字の大きさは，英数字文字集合の文字の大きさよりも大きいか，等しいとする.

⑷ 1バイトあたりのビット数はコンパイラ作成者定義とする.

⑶により，漢字の文字の大きさが2バイト文字（通常の漢字など）と3バイト

文字（JIS補助漢字などに適用）で構成されるEUC符号系をCOBOLで採用するときは，3バイト文字のプログラム中の内部表現を2バイトに圧縮するか，すべての文字を3バイトの内部表現にするかなどの必要が生じる．(3)により，国別文字集合としてシングルオクテットの欧州各国の文字集合が採用可能となる．

2.4.1.4　文字の種類とCOBOL文字集合

実行時に使用可能な文字とは，コンパイラ作成者が決めた文字集合に含まれる文字すべてである．これらの文字は，ソースプログラム上の注記や定数の中でも使用できるものとする．ソースプログラム上で，定数と注記以外の部分に書ける文字をCOBOL文字集合という．今回は，COBOL文字集合に拡張文字を追加した．

(1) 基本英文字　A，B，～，Zの26文字の英大文字と，a，b，～，zの26文字の英小文字
(2) 基本数字　0，1，～，9の10個の数字
(3) 基本特殊文字　次の19文字　　空白　＋－＊／＝＄，．；（）<>＆：・“‘
(4) 拡張文字：上記以外の文字（漢字，ギリシャ文字などを想定．コンパイラ作成者が定義する．）

ソースプログラム上に書く基本英文字，数字，基本特殊文字は，英数字文字集合の文字でも国別文字集合の文字でもよいとする．例えば，英数字文字集合（半角）で「MOVE　A　TO　B」と書いても，国別文字集合（全角）で「ＭＯＶＥ　Ａ　ＴＯ　Ｂ」と書いても同じである．さらに，同じ文字ならば，それが英数字文字集合の文字であろうと，国別文字集合の文字であろうと，定数と注記以外では等価とした．英小文字と英大文字も同じ文字ならば等価とみなす．

漢字などの拡張文字は，利用者定義語にも使用できるから，データ名，ファイル名，プログラム名などに使ってもよい．しかし，構文そのものが国際化されているわけではないので，MOVEといった動詞を漢字で表現することはできない．

例：IF　預金額　＜　引き落とし額
　　MOVE　払い出し不可フラグ　TO　処理結果.

2.4.1.5　自由書式

国別文字の実装のほとんどは2バイト以上なので，今までのCOBOLの固定書式（1～6カラム，7カラム，8～12カラム，最右端カラム（通常72カラム）以降のそれぞれに書く事項が決められていた）では書きにくい．このため，カラムを意識しない自由書式を追加した．

2.4.2　国別文字のデータ定義と操作

2.4.2.1　国別文字項目

COBOLでは，データ型に対応する用語をデータ項目という．既存のデータ項目の種類は，英字項目，英数字項目，英数字編集項目，数字項目，数字編集項目，ビット項目などである．いわゆる文字列型に対応するものは英数字項目であり，英数字文字集合のどんな文字でも収容可能である．英数字項目の大きさはPICTURE句で一意に決まるので, C言語のように文字列の最後を示す特別なコードは不要である．この英数字項目に対応する国別文字項目を，国別文字集合の文字を収容するデータ項目として追加した．

COBOLのデータ項目は，PICTURE句とUSAGE句を使って指定する．PICTURE句は，文字の種類を指定し，英字はA，数字は9，英数字文字集合の任意の文字はXと書く．国別文字集合の任意の文字はNと書くこととする．データ項目の内部表現（2進数，パック10進数，浮動小数点，英数字文字集合，各国文字集合，ブール，ポインタなど）はUSAGE句で指定する．各国文字項目に対するUSAGE句はNATIONALと書くか，省略する．

2.4.2.2　数字項目の内部表現

PICTURE句とUSAGE句とは別に指定する文法なので，数字項目，数字編集項目およびブール項目のUSAGE句にもNATIONALと指定可能にし，この場合内部表現として国別文字集合が使われることになる．このように内部表現を多様に指定できるのがCOBOLの特長である．

例： 01 成績表.

02 氏名 PIC N(10) USAGE NATIONAL.

02 国語 PIC 9(3).

02 数学 PIC 9(3).

この例では，氏名には国別文字集合の文字が入り，国語，数学の点数として英数字文字集合の数字が入る．国語，数学の点数の内部表現として国別文字集合の数字を入れたいときは，次の例のように各項目に“ USAGE NATIONAL ”を追加すれば良い．これらは国別文字項目ではなくて，あくまでも数字項目なので，算術式中に記述可能な項目である．

例： 01 成績表.

02 氏名 PIC N(10) USAGE NATIONAL.

02 国語 PIC 9(3) USAGE NATIONAL.

02 数学 PIC 9(3) USAGE NATIONAL.

2.4.2.3 国別文字定数

英数字定数には, 英数字文字集合の任意の文字を使用できる. これと同様に, 国別文字定数を新設し，国別文字集合の任意の文字を使用できることとする．この二つの定数は，先頭にNがあるかないかで区別する．

例：01 項目1 PIC X(8) VALUE “ ABC ”.

01 項目2 PIC N(3) VALUE N“ 成績表 ” .

項目1は英数字項目であり，初期値はVALUE句で英数字定数を設定できる．項目1の初期値として”成績表”のような国別文字が書けるかどうかは, 処理系依存, すなわちコンパイラ作成者が決めてよい事項とした.「成績表」のように一見3文字の国別文字であっても，英数字文字集合と国別文字集合の間に1けたのシフトコードを必要とする処理系の場合は，先頭と最後にシフトコード用の1けたが確保され8バイトの定数となり，シフトコードが不要な処理系の場合は，6バイトの定数となる．このように，英数字定数に国別文字を許すと，処理系によって定数の大きさが異なる可能性があり互換性が保証できないので，記述可否を処理系依存としている．これに対して，項目2では，シフトコードのない状態の国別文字3けたの成績表が初期値となる．

2.4.2.4　国別文字項目の操作

国別文字項目の転記や比較などのデータ操作は，英数字項目が扱えるMOVE，IF，EVALUATE，STRING，UNSTRING，INSPECT，INITIALIZEなどすべての文で可能とし，構文は英数字項目の構文と同一とした．利用者が英数字項目のデータを国別文字項目に変更する場合には，定義だけを変更すればよく，操作（文）の変更は不要である．

2.4.2.5　関数

次の関数を追加し，英数字文字集合と国別文字集合との間で同じ文字の内部表現の変換を可能とした．対応する文字がない場合は，関数のオペランドで指定する代替文字が設定される．

⑴ NATIONAL-OF関数：英数字文字集合の文字（列）に対応する国別文字集合の文字（列）を求める

⑵ DISPLAY-OF関数：NATIONAL-OF関数の逆．

なお，文字の大きさや符号系を意識する関数については，新しい関数を用意した．

⑴ BYTE-LENGTH関数：既存のLENGTH関数で国別文字項目を指定すると，文字数が求まる．BYTE-LENGTH関数は，バイト数を返す．

⑵ CHAR-NATIONAL関数：指定した国別文字の国別文字集合中のプログラム照合順序番号を求める．英数字文字集合のCHAR関数に対応している．

2.4.3　ファイル処理

2.4.3.1　シフトコードの自動挿入・削除

国別文字集合と英数字文字集合をシフトコードで切り替えるシステムでは，レコードをWRITE文で印字する場合に，シフトコードを挿入する必要がある．自動挿入の指示は，ファイル記述項にFORMAT CHARACTER句を書いて行う．

COBOLで国別文字をディスプレイ画面との間で入出力したりするときの画面機能は，たいていCOBOLの外部にあり，シフトコードの存在を意識する必

要はない．また，COBOLで扱う外部ファイルでは，項目ごとに属性（データ項目や大きさ）が確定しているので，シフトコードを保存する必要はない．このため，ファイル入力時にシフトコードを自動削除する機能を支援している実装はほとんどなかった．今回の言語仕様では，出力と入力で機能が対称性をもつことを重視し，データ部のファイル記述項にFORMAT CHARACTER句を書くと，READ文によるファイル入力時にシフトコードを自動的に削除することにした．その概念を**図2.4**に示す．ここで，Sはシフトコードである。

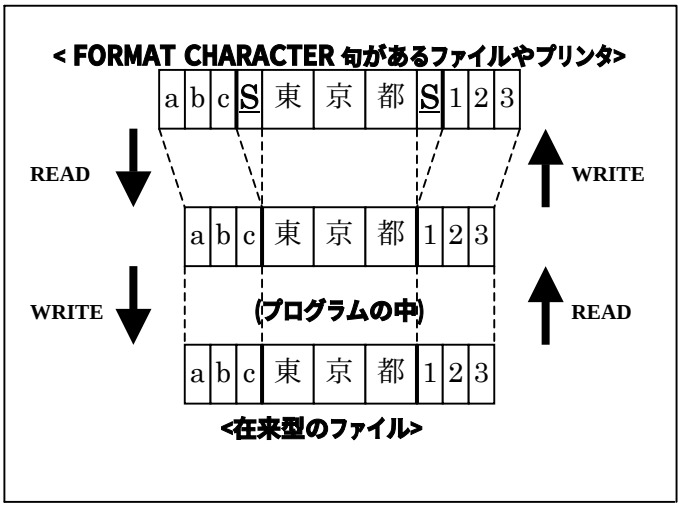


図2.4　シフトコードの自動挿入・削除

2.4.3.2　相対的カラム番号

プリンタによる定型帳票作成のための報告書作成（report　writer）機能では，従来からCOLUMN句があり，左端を1文字目とする絶対番号で出力項目の印字位置を指定してきた．印字位置の絶対指定は，1文字の印字幅が同じであることが暗黙的に仮定できた英数字の世界では不便はなかったが，漢字など1文字当たりの印字幅が大きい文字が混在すると困る．印字位置の意味は，出力文字の番号なのかバイト数なのか不明確となってしまった．このCOLUMN句に，直前の項目からの相対印字位置指定を追加した．

今回の言語仕様では，基本的な印字幅は英数字文字集合の文字の幅と一対一対応とし，国別文字集合の文字幅はその整数倍（整数値はコンパイラ作成者定

義）と規定した．このように規定しても，国別文字集合の文字幅がコンパイラの実装により異なる場合には，絶対番号指定のCOLUMN句では互換性がなくなるので，相対的な印字位置を導入した．これで，国別文字集合の文字の大きさが異なるコンパイラ間の互換性が保証された．

2.4.3.3 符号系が異なるファイルの入出力

現行の規格，すなわち英数字文字集合の世界では，プログラム中で用いる符号系とファイルの符号系とが異なっていても，入出力可能である．ファイル入出力時に符号変換を行うように指定すると，たとえば，メインフレームで作成したEBCDIC符号系ファイルをASCII符号系のUNIXシステムで容易に操作できる．このとき，プログラム内部で参照するレコード領域にはASCII符号系の文字が入る．

これと同じことを，国別文字集合に対しても可能とした．これにより，たとえば，符号系がASCIIとシフトJISのPCシステムで，メインフレームで作成したEBCDICとベンダ固有漢字符号系との混在ファイルが取扱い可能となる．符号変換は入出力時に自動的に行われる．

プログラム内部とファイルの符号系が異なることは，自動変換でさばくにしても，データの比較やSORT文でのソートキーの比較はファイルの符号系の照合順序で行いたいことがある．このために，照合順序の変更をSET文で指定可能とした．

2.4.4 ロケール

ロケールは，「国家，文化及び言語の地域規約」として定義される．1968年の最初のCOBOL規格から，国・地域により異なる表現をサポートする機能があった．DECIMAL POINT句による小数点文字（.）と数値けた区切り文字（,）の切替機能と，CURREENCY SIGN句による通貨記号の指定機能である．

これに加え，次期COBOL国際規格には，C言語のロケール（地域固有仕様）対応機能を追加することにした．COBOLのロケールは，LC-COLLATE（照合順序），LC-CTYPE（文字の字類分けと大文字・小文字の変換規則），

LC-MESSAGE（メッセージの形式），LC-MONETARY（金額の表示形式），LC-NUMERIC（数値の表示形式），LC-TIME（日付と時刻の表示形式）の六つであり，環境部でどのロケールを使用するかを指定する．ロケールの値は，SET文で実行時に変更できる．

2.5　国際化機能の有効性と実装方式

2.5.1　応用例とその有効性

2.5.1.1　応用例

国際化機能を使用した COBOL プログラムの例を**図 2.6** に示す．このプログラムはメインフレームで作成した成績表ファイル（符号系は EBCDIC と JIS 漢字）を入力し，PC で成績表（**図 2.5**）をプリンタに印刷する．プログラム中の下線箇所は，国際化機能の利用を示す．次に若干の説明をする．

1 行目の行末など： ピリオドは読みやすいように英数字文字集合（半角）ではなく国別文字集合（全角）の文字を使用している．なお，どちらを使っても機能は同じである．

2 行目など： 利用者定義語（プログラム名，符号系名，ファイル名，データ名および手続き名）に国別文字（日本語の文字）を使用しており，日本語を母国語とする者には分かりやすい．

8 行目：“ JIS 漢字コード ”という名前で国別文字集合符号系を指定している．これは，16 行目の code-set 句で参照されており，入力ファイルの符号系が JIS 漢字コードであり，実行時に符号系変換が必要なことを指定している．

成績表

氏名	国語	数学	英語	合計
千代田一郎	100	85	90	275
港和子	87	100	94	271
品川創造	90	68	76	234
：	：	：	：	：
：	：	：	：	：

図 2.5　図 2.6 のプログラムの印字例

```
01  identification division.
02  program-id.   国際化機能使用例.
03  environment division.
04  configuration section.
05  special-names.
06   alphabet.
07     EBCDIC コード for   alphanumeric is EBCDIC
08     JIS 漢字コード   for   national    is   JIS-KANJI.
09  input-output section.
10  file-control.
11     select 成績ファイル assign to “sys001”.
12     select 成績表          assign to system-printer.
13  data division.
14  file section.
15  fd   成績ファイル
16     code-set is EBCIDIC コード,  JIS 漢字コード   format character.
17  01 成績レコード.
18     05   氏名          picture    n(8).
19     05   国語          picture    9(3).
20     05   数学          picture    9(3).
21     05   英語          picture    9(3).
22  fd   成績表      format character.
23  working-storage   section.
24  01 タイトル１ value n”                          成績表”.
25  01 タイトル２ value n”氏名                      国語   数学   英語   合計”.
26  01   個人成績.
27     05   氏名     picture    n(8).
28     05            picture    x(3)   value space.
29     05   国語     picture    zz9.
30     05            picture    x(3)   value space.
31     05   数学     picture    zz9.
32     05            picture    x(3)   value space.
33     05   英語     picture    zz9.
34     05            picture    x(3)   value space.
35     05   合計     picture    zz9.
36  procedure division.
37     open   input 成績ファイル  output  成績表.
38     write   file   成績表   from   タイトル１.
39     write   file   成績表   from   タイトル 2.
40  入力.
41     read 成績ファイル  at end
42       close 成績ファイル   成績表
43       stop   run.
44       move corresponding 成績レコード  to  個人成績.
45       compute  合計 =   国語  of 成績ファイル
46                      ＋数学  of   成績ファイル＋  英語  of 成績ファイル.
47       write   file   成績表   from   個人成績.
48       go   to   入力.
```

図 2.6　国際化機能プログラム例

16 行目と 22 行目： format character 句でファイル入出力時にシフトコードの自動挿入削除を指定している．

18 行目など： picture 句のｎで国別文字項目を指定している．

24 行目など： 初期値として n”…”の国別文字定数を指定している．

37 行目以降：文には下線がつけていない．手続き部の文では国別文字項目を転記したり，入出力したりしているが，構文は英数字項目を扱うときと同一である．

2.5.1.2 有効性

プログラム例からも明らかなように，国際化機能を使うことにより，利用者およびプログラマにとって多くの点で便利になった．その有効性を次にまとめて列挙する．とくに(1)(a)と(2)(a)(b)の効果が大きい．

(1) ほかのプログラム言語でも実現していること．

(a) 利用者インターフェース（画面や帳票）やデータに国別文字（母国語の文字）を使えるようになったこと．

(b) 入出力時に，シフトコードを意識しなくてもよくなったこと．

(2) ほかの標準プログラム言語で実現していないこと．

(a) 利用者定義語に国別文字，すなわち母国語を使えるようになったこと．母国語の仕様書と英数字のプログラムとの間での用語（データ項目名，手続き名など）の変換が不要になった．これにより，仕様記述で使っている用語と同じ用語をそのままプログラム中で使用可能になり，生産性が向上するとともに，変換ミスが入りこむ余地がなくなり信頼性も向上する．さらに，プログラムの可読性が向上して，保守要員が変わってもプログラムの内容理解が容易になるので，要員確保に管理者が悩まなくてもすむようになった．筆者の経験したプロジェクトでは，約 50 万行の COBOL プログラムの利用者定義語に日本語を使用し，開発時にそれまでのプロジェクトと比べ不良頻度が約 2/3 で済み，その後の保守要員を急激に削減でき，今は 1 人で済ますことができている．標準プログラム言語で，このように利用者定義語に国別文字を使えるようにしたのは，COBOL が最初であり， C や Fortran の規格ではまだ許していない．

1998 年にやっと C++の規格で許すようになったが，商用での実装例はまだない模様である．

(b) 手続き部の文や関数の書き方が国別文字でも英数字項目でも同一であること．国別文字項目の picture 句あるいは国別文字定数だけ書きなおせば, 英語用や中国語用など各国で実行する共通プログラムを作成可能である．**図 2.6** のプログラムを英語用に修正するには，①8 行目と 16 行目の JIS 漢字コードの定義と参照を削除し，②18 行目と 27 行目の picture 句の n(8)を x(16)に変更し, ③24 行目と 25 行目の国別文字定数の内容を英語にし英数字定数に変更すればよい. 中国語用に修正するには，①8 行目と 16 行目の JIS 漢字コードを中国語コードに変更し，②24 行目と 25 行目の国別文字定数の内容を中国語に変更すればよい. このように，まず “ 国際化プログラム ” を作成し，それを基にロジックの変更をせずに特定の定義だけ変更すれば, 各国向けのプログラムが作成できるようになった. このような国際化プログラム作成の必要性は, 多くの国で流通する汎用アプリケーションパッケージやグローバル企業の業務プログラムで切実なものになっており, それらで最も利用されているのが COBOL 言語なので，開発者にとって COBOL 国際化機能は有効である． なお, C や C++では文字型とワイド文字型とは関数名が異なるので，ロジック中のコーディングの修正が必要となることは, すでに述べた.

2.6 今後の課題

COBOL次期国際規格の国際化機能で解決できなかった主な課題は次の四つで，これらは今後の規格改訂時に検討していくべきである.

2.6.1 英数字と国別文字の混在

住所表記において県，市，区，町，村などの名前は国別文字を用い，番地は1バイトの数字とハイフンで表現すると，印字や表示幅が短くてすみ，便利である.

しかし，現規格には国別文字集合と英数字文字集合が混在可能なデータ型はない．従来からの利用者は，これをやむなく英数字項目に格納しているが，シフトコードの処理が厄介であり，項目の最後に存在する国別文字の後半の1バイトが収容できずバイト落ちし文字化けするなどの現象に悩まされている．

これらの面倒が起きないようなデータ型，すなわちファイルや端末からの入力処理，プログラム内処理，ファイル，端末，帳票への出力処理で整合性があり安心して使えるような混在データ型の切実な新設要求があるが，この要求に対して整合性のある普遍的なデータ型はまだだれも実現できていないし，国際規格の国際化機能でも実現できなかった．

2.6.2 符号系の制約と多国語環境

言語仕様の簡潔性と利用の便利さからは，一つのマルチオクテット文字符号系だけを想定し，データ型として1種類の文字型だけですますのが，理想的である．

しかし，コンピュータが生まれてから20年以上の長い間，1バイトの英数字だけしか文字として扱えなかったという制約から，1バイトの符号系とマルチオクテット文字の符号系の混在が，現実となっている．さらに，現在使用されているマルチオクテット文字符号系は，2オクテット系がほとんどすべてであるが，これも今後の歴史的な制約となりうる．2オクテットでは，高々65ｋ個の文字しか割り当てできないので，中国の漢字と日本の漢字を同時に使うことは，難しいことがよく知られている．

インターネットの普及により，同時に複数の地域のマルチオクテット文字を使用したいという多国語環境の必要性が高まっており，その研究も進んでいる[片岡97]，[西垣2001]．FortranやSQLのように言語によっては，同時に複数個のマルチオクテット文字集合をサポートできるものがあるが，ほとんどの言語は，一つのマルチオクテット文字集合しか使用できない．できても，文字集合のロケールを切り替えることまでであり，複数を同時に使用できない．

将来の多国語環境を実現する符号系とOSがどうなるかによって，プログラム言語の多国語環境仕様も異なってくるが，一つの符号系ですべてを済ますという理想的なものは無理である．プログラム言語の文字型として，1バイトのも

の，今現実にある2バイトのもの以外に，多国語環境用のより大きな文字型（おそらく4バイト）が必要になるだろう．これはCOBOLに限らず，2005年以降のプログラム言語の規格改訂時には検討が必要になると考えられる．

2.6.3 システムモデルの限界と環境の複雑化

COBOL国際化機能は，1990年頃のシステム環境であるスタンドアロンシステムを前提としている．そこでは，一つのOSが全体の環境を制御しており，システムのロケールを一つ定めれば，それによって配下のプログラムなどの環境が自動的に決まり，あとは，必要に応じてロケールを切り替えればよかった．

現実のコンピュータ環境は，ネットワークで結ばれた複数のコンピュータがそれぞれ自律的に動作しながら，ときには協力して動くというように，複雑化している．たとえば分散オブジェクトの世界では，起動したオブジェクトが自分のコンピュータで動いているのか，他のコンピュータで動いているのかは分からない．このような中では，それぞれのロケールがどうなっているかはアプリケーション側では予想できない．単純な例として，EUC符号系のデータサーバ，シフトJIS符号系のアプリケーションサーバ，10646符号系のクライアントの3階層システムを考えただけでも，その難しさは容易に理解できよう．

このような複雑な環境で国際化の機能をどう実現していくか，まずシステムモデルの再構築からはじめなければならない大きな課題である．

2.6.4 文字の照合順序

マルチオクテット文字の照合順序は複雑である．符号系の文字の符号化順序とは別に，英語辞書のようにAとaは隣接した位置におく方式，日本語の国語辞書のようにあいうえお順（漢字は読みの順）の方式，漢字辞書のように部首順の方式など，現実の世界では多くの照合順序が必要である．

COBOL言語では，英数字文字集合と各国語文字集合のそれぞれに，利用者が照合順序を指定できる仕様が入っている．照合順序が重要なデータベースでは，SQL3（SQL99）でよりきめ細かな照合順序の仕様が検討され，1999年に国際規格として制定された．データベースとプログラム言語との親和性は重要

なので，SQL3の仕様を参考にしてより本格的な照合順序仕様をプログラム言語でも検討すべきであろう．

2.7 まとめ

本章では，COBOL国際化機能の開発と国際規格化提案について議論しその有効性を示した．従来は国際規格といっても英文字26字しか使用できなかったが，この国際機能により，日本，中国，韓国，タイなどの主要なマルチオクテット文字使用の国・地域で適用可能なだけでなくて，ヨーロッパ各国にも適用可能な汎用的な枠組み各国文字の使用を実現できた．さらに，文などのロジック部分を変更することなく，各国文字を定義する部分だけ変更すれば現地語化可能な国際化プログラムの作成が可能となった．国際化機能を用いると母国語で利用者定義語を定義できるので，プログラムの生産性，信頼性，保守性が向上する．これが実用化されているのもCOBOL言語だけである．

国際化機能を含む COBOL 国際規格改訂は，2002 年に予定されている．国際化機能については，審議は完了し，仕様は確定している．さらに，主要部分は各ベンダが実装済みで利用者の使用は各国で進んでおり，着実に国際化機能の有効性が利用者に享受されてきている．21 世紀の実用に耐えることのできる国際化機能が日本の提案により実現したと評価できる．

第 3 章　分かち書きの日本語プログラム言語
“ EAGLE/4GL ”
―― オンライン業務プログラムの環境独立処理方式――

3.1　はじめに

企業や官公庁の基幹業務（経理，人事，生産管理，販売，銀行取引など）のシステム開発では，COBOL や PL/I の第 3 世代言語を使い，EAGLE や SEWB などのシステム開発支援ツール（CASE ツール）を用いて生産性向上を図ってきた [葉木 84]，[葉木 86]，[津田 92]．このツール開発には筆者も参加し，開発方式と適用効果については大野の一連の論文で報告している[大野治 95]，[大野治 98]，[大野治 2000]，[大野治 2001a]，[大野治 2001b]．この開発方式の対象業務の主要稼働環境は大型メインフレームであったが，オープン環境に対しても有効である．

別の開発方式として，プログラム言語の水準を上げるやり方がある．この言語は，簡易言語とかエンドユーザ言語[酒井 84]といわれる分野であり，1980 年代半ばから 1990 年代の前半にかけて第 4 世代言語[Martin85]という名前で呼ばれ，各ベンダが競って製品化した[古宮 90]．この言語の対象業務は企業の基幹業務であるが，稼動環境は中小型メインフレームやオフィスコンピュータが中心で，大企業の部門や中小企業の情報システム部門と，フルターンキーのディーラーが主な利用者だった．この稼動環境はオープン化の波に流され，現在は市場の表舞台では主役ではなくなった．しかし，その設計思想は貴重であり，現在のオープン環境にも適用可能な普遍的なものも多い．本章では，その一つとして “ 環境独立なオンライン業務プログラム処理方式 ” を中心に紹介する[今城 2001b]．

本章の構成は次のとおりである．**3.2** では，1988 年に出荷された中小型メインフレーム向けの第 4 世代言語 EAGLE/4GL の開発背景，特長および機能を述べる[脇坂 89]，[今城 89]，[今城 90a]．**3.3** では，1990 年出荷の EAGLE/4GL 後継製品において，その実行環境を大型メインフレーム，オフィスコンピュータ，UNIX に広げ，開発環境を UNIX としたことにより必然的に必要となった拡張項目を論じる[吉野 91]，[西尾 93]，[吉野 93]．**3.4** では，そこで採用した

“環境独立オンライン業務プログラム処理方式”について述べる．また，この方式がオープン環境でも適用可能なことを示す．3.5.では，この方式の今後の課題を明らかにし，3.6.では，本章のまとめを述べる．

3.2 第4世代言語EAGLE/4GL

3.2.1 開発の背景

一般的に，第4世代言語は次のように定義されている．

(1) ユーザフレンドリである．
(2) 専門家でないプログラマでも結果が得られる．
(3) COBOL（第3世代言語）の1/10の命令数で結果が得られる．
(4) データベースシステムを直接利用できる．
(5) プロトタイプを手早く作成し，変更することができる．
(6) 一般的なユーザが2日間の学習で一応使えるようになる．
(7) COBOLの1/10以下の期間で結果が得られる，など．

現実に市販された第4世代言語がこれらの項目にすべて合致していることは少なく，いずれかに重点を置いている．

第4世代言語を大別すると，非定形的な業務を利用部門で直接作成するためのエンドユーザ言語と，コンピュータの専門家（情報処理部門）が基幹業務を開発するための高生産性言語と呼ばれるものの2種類に分かれる．

第4世代言語EAGLE/4GLは後者の“基幹業務開発用の高生産性言語”を目的としている．この理由は，この言語の最初の実行・開発環境が中小型メインフレームであったことが大きい．中小型メインフレームでも比較的大きな機種のユーザにおいては，そこの基幹業務は数人～10数人の情報処理部門で開発されており，基幹業務はそれなりに難しいためエンドユーザに任せることもできず，多くのバックログをかかえていた．ここでは，一にも二にもシステム開発の抜本的な生産性向上が緊急課題であった．一方，比較的小さな機種のユーザにおいては，専門の情報部門があっても1～2人など少人数のため，自ら基幹業務を開発する余力がなく，ディーラーなど外注会社に全面的に開発を委託することが多かった．この外注金額もそう大きくはなく，競争も激しいので，外注会社にとってシステム生産性向上が最大命題だった．この2つの要求に応える言語

として，EAGLE/4GLが開発された．

EAGLE/4GLの開発時に，マーケティング部門が開発部門に要求した生産性向上の目標値は，COBOLの3～4倍だった．この数値の根拠は，EAGLE/4GLが稼動する中小型メインフレームの新機種の販売目標に対し，それを主に販売するディーラーのフルターンキーに従事するSEの数が少なく，販売目標台数を達成するには当時のSEの生産性と比べ3～4倍の向上は必要という経営上の要求による．この値は先に示した第4世代言語の定義には至っていないが，他社協業上“第4世代言語”という分野の製品も必須だったため，あえて製品名に“第4世代言語（4GL）”の名前をつけ，10倍という数値は努力目標とした．4GLの前にEAGLEをつけたのは，日立の開発支援ツールの総称がEAGLEであったことによる．

3.2.2 特長と機能

EAGLE/4GLの特長は次の6つである．

(1) 対話処理によるシステム開発

EAGLE/4GLの開発時の動作環境は，日立の中小型メインフレームのOSであるVOSKの対話環境である[矢島89]．VOSKの対話環境は，大型OSのTSS環境と類似しているが，OS本体にデータベース機能を持たせたことにより，TSSとは別環境だったオンライン業務環境をも包含している．この対話環境で日本語メニューの開発環境画面（**図3.1**）により，システ設計から運用・保守までの一連の作業が可能である．海外用には英語メニューも用意している．

図3.1 EAGLE/4GLによる業務選択メニュー 目的業務の番号を選択することで業務を開始できる.

(2) プロトタイピングによる画面・帳票定義

図3.2に示すように,業務画面・帳票イメージをフルスクリーン画面上でラフスケッチ定義することにより,直ちに確認画面を表示するプロトタイピング技法を採用している.これにより,システム設計の段階で最終的な画面・帳票様式,大きさ,色などの詳細な仕様確認が可能であり,早期に入出力フォーマットが確定できる.

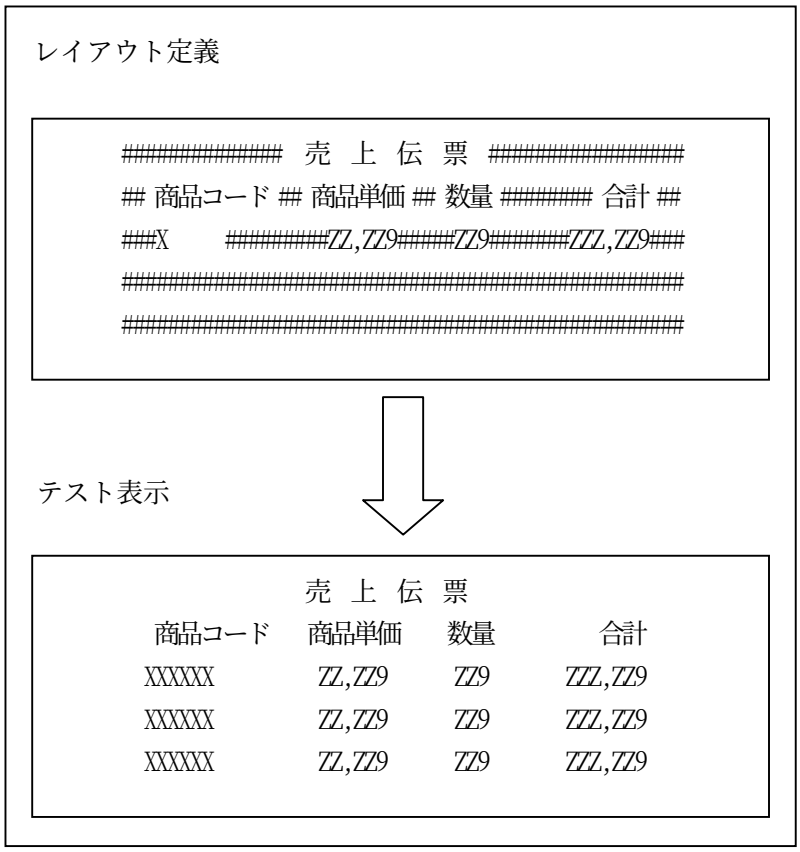


図3.2　プロトタイピングによる画面・帳票定義　　画面・帳票の入出力項目を指定するだけで，それらのイメージをその場で確認できる．

(3)　処理のパターン化

EAGLE/4GLでは，大型メインフレームのシステム開発支援ツールEAGLEのバッチ22パターンとオンライン5パターンを参考にし，**図3.3**に示すようにコンピュータ業務処理を3つのパターンに集約した．

(a) 対話パターン

(b) ファイル作成・更新パターン（バッチ）

(c) マッチングパターン（バッチ）

このパターンと一致する業務処理については，基本アルゴリズムを考えることなく,処理構成メニューの空きを埋めていくだけでプログラムが作成できる．

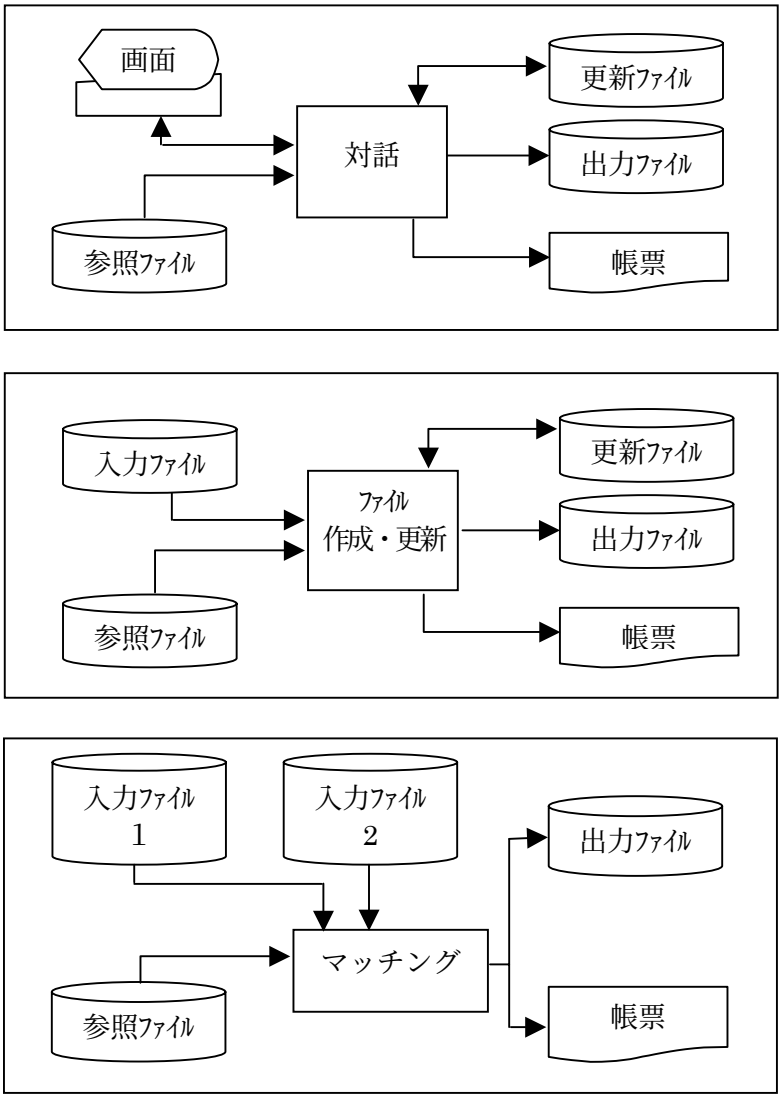


図3.3　EAGLE/4GLの処理パターン　　対話１パターン，バッチ2パターンでプログラムが組める．

各パターンで可能な処理の種類は以下の通りである．

(a) 対話パターン

①務振り分け，②問い合わせ，③ファイル更新，④ファイル更新と帳票出力，⑤データエントリ

(b) ファイル作成・更新パターン（バッチ）

①ファイルの分配と抽出，②ファイルの集約と照合，③ファイルの更新レ

コード追加，④帳票出力，⑤「ファイルを更新しながら帳票出力する」などの複合形

(c) マッチングパターン（バッチ）

①ファイル2本の照合（1：N），②ファイル照合時のエラーリストの出力処理

(4) 日本語プログラミング

3.4の上部のメニューは処理構成定義で，3つのパターンごとに異なる図柄で表示され，そこにはプログラムの入出力（ファイル，画面，帳票など）の仕様書名を投入する．この処理構成定義から**図3.4**の下部に示すEAGLE/4GLの処理仕様書（ソースプログラム）が自動生成される．EAGLE/4GLを使用する設計者（プログラマ）は，この自動生成された仕様書を基にエディタを用いて業務仕様を追加する．

この追加のために，各種の記述文や関数を用意している．これらの処理記述文（転記，四則演算，条件により2分岐（if文に相当），条件により多分岐（case文に相当），ループ（while文やfor文に相当）など）と関数名は日本語である．画面・帳票・DB・ファイル・データの項目名称にも日本語が書けるので，英語をベースとした第3世代言語とくらべ，日本語を使う者にとって親和性が強く読みやすく書きやすい．なお，日本語以外に英語を基調とした構文も用意している．

図3.5にファイル作成・更新パターンのプログラム例を示す．

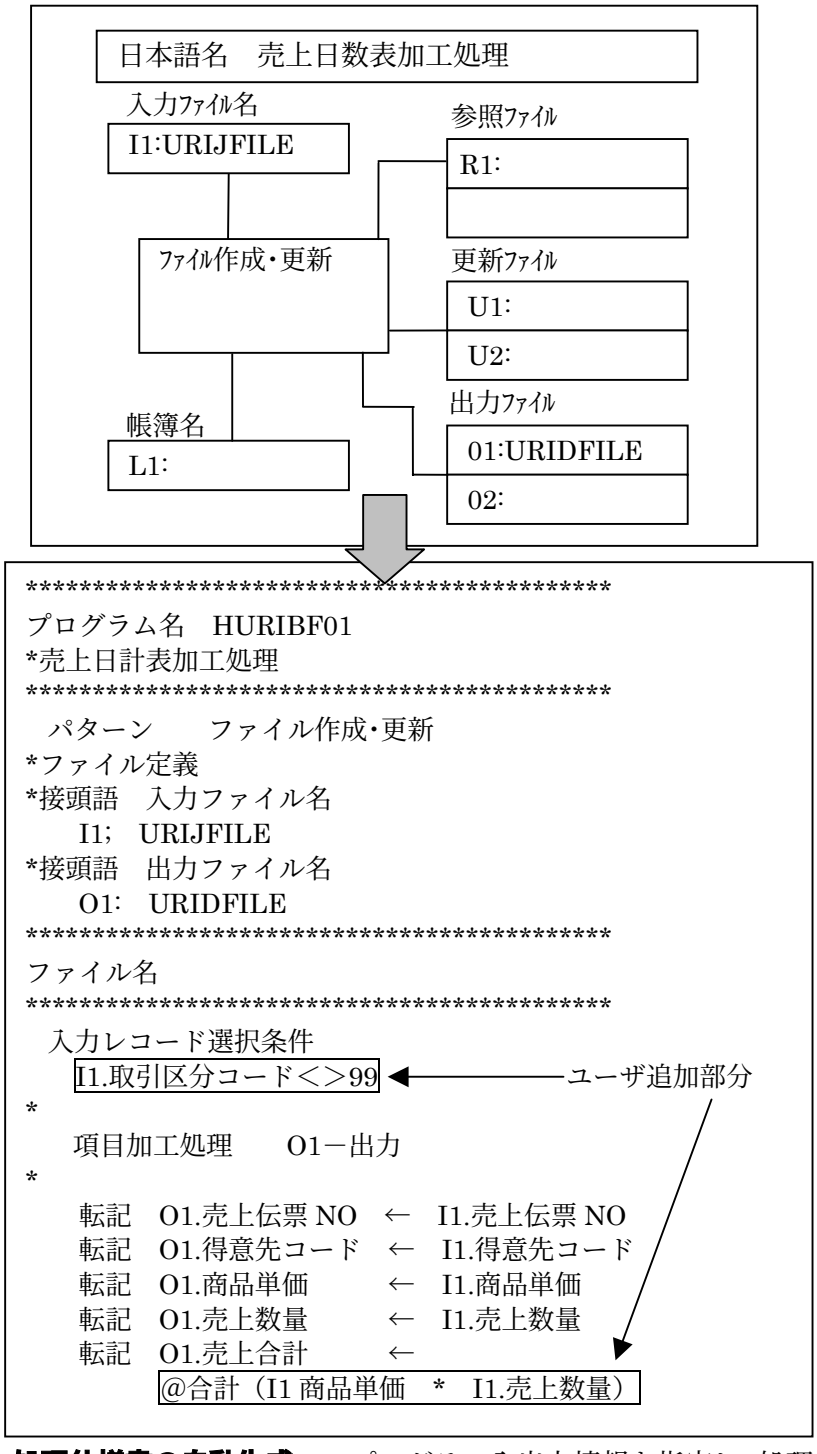


図3.4　処理仕様書の自動生成　　プログラム入出力情報を指定し, 処理仕様書を自動生成する．データ項目の名称や処理の指示（文）に日本語が使える．

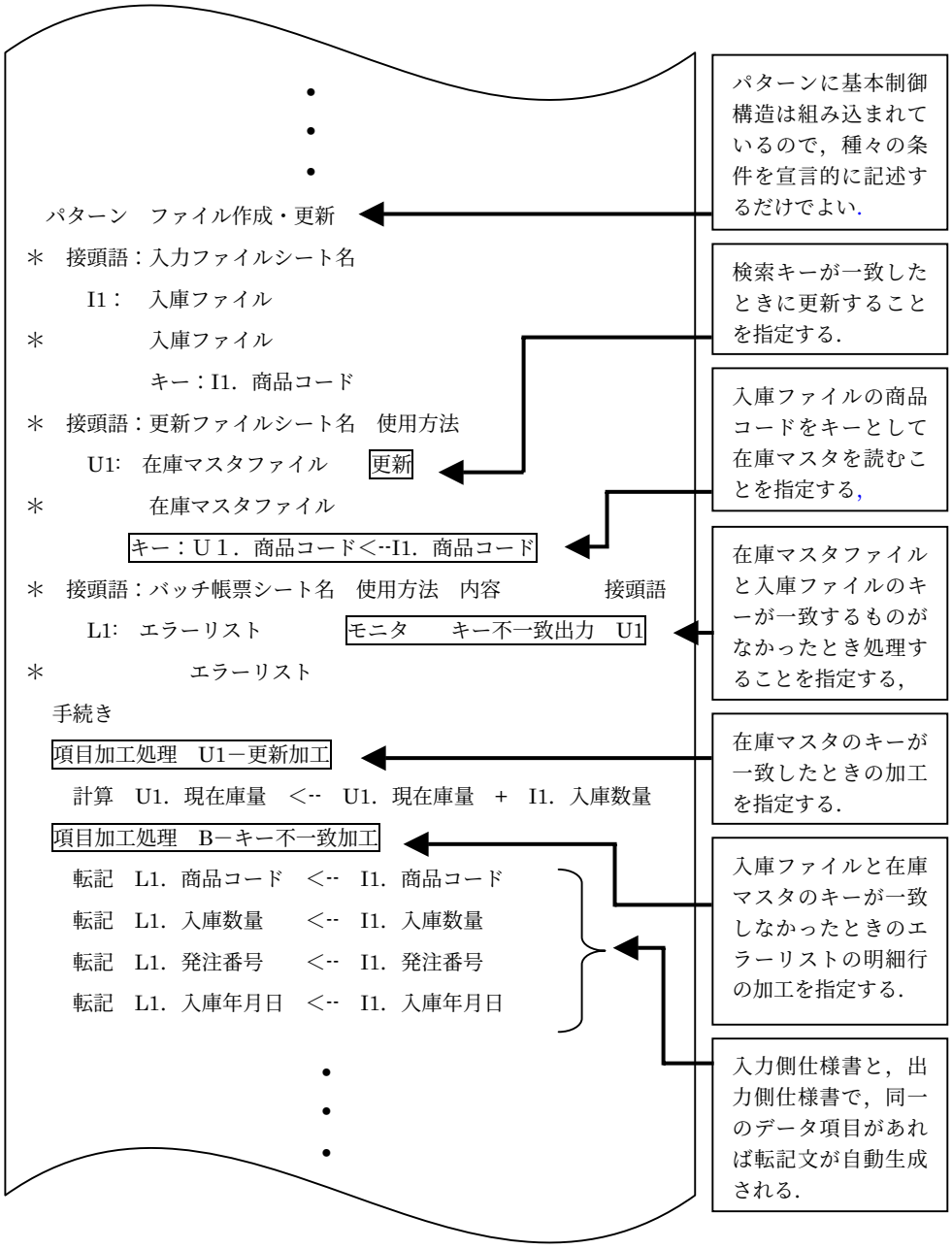


図3.5　EAGLE/4GL処理仕様書の例　EAGLE/4GL仕様書は，日本語記述によって保守効率向上を図っている．（[吉野93]より）

(5)　簡易仕様自動定義（自動作表）機能による業務仕様自動生成

帳票を出力するプログラムの開発に有効な機能であり，入出力構成，帳票の形，および項目情報を記述するだけで，帳票出力プログラムを完全自動生成可能とした．すなわち，プログラミングレスの実現である．これを適用できるのは，バッチ帳票の中でもよく使われるファイル内容を直接印字する一覧表や，小計や合計を印字するような集計表であり，**図3.6**に示す入出力構成が条件である．

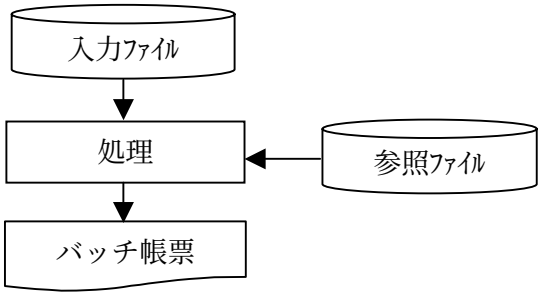


図3.6　簡易仕様定義（自動作表）入出力構成　　この入出力構成であれば，簡易仕様定義（自動作表）による開発が行える．

(6)　COBOLソースとXMAP定義文を生成

新しい言語の開発において次の4つの大きな作業が必要である．

①言語仕様の開発，②コンパイラの開発，

③実行時ライブラリの開発，④開発環境の開発

言語を普及するためには実行時の性能と品質も重要である．すなわち，②で生成するオブジェクトの効率を良くし，③では性能と品質に特に配慮する必要がある．これらの開発負荷を低減するため，EAGLE/4GLではCOBOLのソースプログラムを生成し，実行時の性能と品質の課題はCOBOLに頼ることにした．このことにより，EAGLE/4GLの開発においては②と③の作業が大幅に低減でき，①と④に注力できた．また，COBOL生成としたことにより，新言語に対するユーザの“将来とも存続するか”という危惧に対しても，“いざとなったらCOBOLで保守可能”という逃げ道を用意できた．

また，EAGLE/4GLでは画面・帳票の実行時ライブラリの開発量も多大なので，EAGLE/4GLの画面・帳票定義から，汎用の画面・帳票製品であるXMAPの定義文を生成するようにし，作業量低減と実行時性能･品質の確保を図った．

3.2.3 生産性実績

EAGLE/4GLの生産性を評価するため，受注・販売管理システムをEAGLE/4GLとCOBOLの両方で記述実験した．

プログラムの内容は，受注データの入力や発注商品一覧表，売上日報印刷処理など9本である．この二つの記述量の比較は次のように行った．

(1) EAGLE/4GLとCOBOLとも同じ入出力構成なので，データ定義部分は同等とみなし．手続き部（procedure division）に記述した行数を比較する．

(2) 注釈行は評価対象外とする．

(3) COBOLは手続き部の行数を記述量として評価する．

(4) EAGLE/4GLではパターンと入出力構成を指定すると，ソースプログラムが生成される．ユーザはこれに対して追加・修正し，プログラムを完成する．この追加･修正があった行数をEAGLE/4GLの記述量として評価する．

評価結果を**表3.1**に示す．**表3.1**項番6の二つの言語の手続き部を，**図3.7**と**図3.8**に示す．COBOLが49行に対し，EAGLE/4GLの追加修正の行数は9行である．この記述実験では，EAGLE/4GLの記述量はCOBOLに比べ，バッチ処理で1/10，対話処理で1/3に減少した．

表3.1　EAGLE/4GLとCOBOLの生産性　　受注・販売管理システムをCOBOLとEAGLE/4GLの両方で記述した結果である．

項番	パターン	手続部記述量		比率
		COBOL	EAGLE/4GL	
1	バッチ	96	11	1/9
2	バッチ	11	0	―
3	バッチ	59	6	1/10
4	バッチ	113	10	1/11
5	バッチ	21	1	1/21
6	バッチ	48	9	1/5
7	対　話	232	72	1/3
8	対　話	197	72	1/3
9	対　話	116	43	1/3

```
009400  PROCEDURE       DIVISION
009500       OPEN INPUT  HSORT-FD
009600            OUTPUT HATCHUU-FD
009700            I-O    LHATCHUU-FD
009800      ACCEPT 日付  FROM DATE
009900      READ HSORT-FD
010000        AT END MOVE ‘１’TO FLAG
010100      END-READ
010200      PERFORM 主処理 THRU 主処理－終り
010300        UNTIL (FLAG=‘１’)
010400      CLOSE HSORT-FD
010500            HATCHUU-FD
010600            LHATCHUU-FD
010700      STOP RUN.
010800*********************     主処理     *********************************
010900  主処理.
011000      MOVE 商品コード OF HSORT-REC
011100                        TO 商品コード OF LHATCHUU-REC
011200      READ LHATCHUU-FD
011300      COMPUTE 仕入数量＝最高在庫数量 OF HSORT-REC
011400                       －現在庫数量   OF HSORT-REC
011500      MOVE 仕入数量 TO 予約在庫数量 OF LHATCHUU-REC
011600      REWRITE LHATCHUU-REC
011700      MOVE 仕入先名 OF LHATCHUU-REC
011800           TO 仕入先名 OF MEISAI-REC
011900      MOVE 仕入先電話番号 TO 電話番号
012000      MOVE 商品名 OF LHATCHUU-REC TO 商品名 TO MEISAI-REC
012100      MOVE 商品コード OF LHATCHUU-REC
012200        TO 商品コード OF MEISAI-REC
012300      MOVE 色 OF HSORT-REC TO 色 OF MEISAI-REC
012400      MOVE 現在庫数量 OF LHATCHUU-REC TO 在庫数量
012500      MOVE 安全在庫数量 OF LHATCHUU-REC
012600        TO 安全在庫数量 OF MEISAI-REC
012700      IF LINE-COUNT > 17
012800        THEN
012900          COMPUTE PAGE-CUT = PAGE-CUT + 1
013000          MOVE    PAGE-CUT TO ページ
013100          WRITE HATCHUU-REC FROM TITLE-SPACE AFTER PAGE
013200          WRITE HATCHUU-REC FROM TITLE-REC1  AFTER 2
013300          WRITE HATCHUU-REC FROM TITLE-REC2  AFTER 4
013400          MOVE 0 TO LINE-COUNT
013500      END-IF
013600      WRITE HATCHUU-REC FROM MEISAI-REC
013700        AFTER 2
013800      COMPUTE LINE-COUNT = LINE-COUNT + 1
013900      READ HSORT-FD
014000        AT END MOVE ’１’TO FLAG
014100      END-READ
014200  主処理-終り.
014300      EXIT
```

図3.7　表3.1項番6のバッチ業務のCOBOL記述

```
003600*************************************************************
003700  手続き
003800*************************************************************
003900*
004000  項目加工処理        U1－更新加工
004100*
004200        計算      U1. 予約在庫数量      <--    I1. 最高在庫数量
004300                                              －   I1 現在庫数量
004400*
004500  項目加工処理        H-キー一致加工
004600*
004700        転記      L1. 日付              <--   @日付
004800        転記      L1. PAGE              <--   @ページ編集
004900*
005000  項目加工処理        B-キー一致加工
005100*
005200        転記      L1. 仕入先名          <--
005300        転記      L1. 電話番号          <--   U1. 仕入先電話番号
005400        転記      商品名                <--   I1. 商品名
005500        転記      色                    <--   I1. 色
005600        転記      L1. 商品名            <--   商品名色
005700        転記      L1. 商品コード        <--   I1. 商品コード
005800        転記      L1. 仕入数量          <--   U1. 予約在庫数量
005900        転記      L1. 在庫数量          <--   I1. 現在在庫量
006000        転記      L1. 安全在庫数量      <--   I1. 安全在庫数量
```

図3.8　表3.1項番6のバッチ業務のEAGLE/4GL記述　　生成されたソースに対して　　で囲んだ部分が追加または修正箇所であり，全部で9行ある．

3.3　EAGLE/4GL実行環境の拡張と開発環境の一本化

3.3.1　多様な実行環境への適用

1990 年代はじめには，第 4 世代言語の必要性，すなわち，この言語によるシステム生産性向上の期待は中小型メインフレームだけでなく，大型メインフレームやオフィスコンピュータ，さらに UNIX などのオープン環境と多様な分野に広まり，そのサポートは緊急課題であった．また，SE は１つの分野の仕事だけを続けることはまれで，複数の分野のシステム開発に携わることが多い．このようなニーズから，マーケティング部門から製品開発部門への要求仕様は，“多くの分野に適用可能な 1 種類の第 4 世代言語で，開発環境も 1 つだけとする”ということであった．

これらの条件をすべて満たすために EAGLE/4GL の開発環境を UNIX に移行し，その開発環境で対象となる実行環境（大型メインフレーム，中小型メインフレーム，オフィスコンピュータ，UNIX などのオープン環境）の COBOL ソースプログラムと画面・帳票・DB・ファイルなどの定義文を生成するようにした．

さらに，UNIX 環境の COBOL コンパイラや XMAP を用いてオブジェクトを生成し，UNIX で実行することにより，これらの単体テストを可能とした．これが，実現できたのは，COBOL，XMAP，OLTP，DB などメインフレームと互換性を有する主要ミドルウェアが順次 UNIX で稼動したことによる．

3.3.2 必然的に必要となった拡張項目

実行環境の拡大と開発環境のUNIX化により，必然的に必要となった主要な拡張項目を次に述べる．

3.3.2.1 開発環境メニューのGUI化

1980年代の終わり頃にUNIX上の標準GUI機能であるMotifの仕様が決まったので，EAGLE/4GLの開発環境メニューもそれを用いた（日本国内でのMotifの最初の利用者）．このため，VOSKのEAGLE/4GLの開発環境メニュー（CUIインターフェース）と比べユーザインターフェースが抜本的に改善された．

ただし，CUIとGUIのインターフェースが全く異なり，UNIX上の記述言語もC言語しか選択の余地がなかったので，VOSKのEAGLE/4GLのコーディング（COBOLとメインフレームのシステム記述言語とで記述）は流用できず，全面的に再設計・リコーディングとなった．

3.3.2.2 メインフレームOLTP環境の業務プログラムサポート

オンライン基幹業務はVOSKでは対話環境で実行可能であったが，大型メインフレームにおいては，対話環境と類似環境であるTSS環境では一部の簡易な業務だけが実行され，多くの業務はオンライン制御プログラム環境（OLTP環

境，メインフレームではDC環境ともいう）で実行される．EAGLE/4GLでは，このOLTP環境の業務プログラムをサポートする必要がある．これについては，3.4で詳しく述べる．

大型メインフレームでOLTP環境が業務に使われる代表的な理由は次の二つである．

(1) 信頼性：TSS環境でDB共用や異常時の回復などすべてユーザ責任であるが，OLTP環境ではそれらの面倒はオンライン制御プログラム（OLTP）がみてくれる．

(2) 資源の有効活用と応答時間の平準化：OLTPは多くの端末からの大量のトランザクションをキュー管理し，限られたシステム資源を有効に利用しながら，応答時間が均一になるように適切にスケジューリングして，業務プログラムを実行する．TSS環境では，端末ごとに1つの空間を占有するのでシステム資源を大量に必要とする．

3.3.2.3 DB（SQL）サポート

VOSKのDBはリレーショナルライクなDBであり，ファイルに対するread/writeインターフェースでアクセスする．一方，大型メインフレームの業務にはDBが必須で，EAGLE/4GL開発当時はSQLが普及段階に入っていた．EAGLE/4GLで.SQL機能を制限せずに，サポートする必要がある．このために，すべてのSQL構文に対応した日本語SQL構文をEAGLE/4GLでサポートした．

3.3.2.4 生成したソースなどの対象環境への転送

UNIX環境のEAGLE/4GLはメインフレームの3種類のOS（VOS3，VOS1，VOSK）およびオフィスコンピュータのOS（MIOS7）に対応したCOBOLソースプログラムや各種の定義文（パラメタ）を生成する必要がある．さらに，それらをファイル転送などで，個々のOS環境に送りコンパイルするなどのしかけをサポートする必要がある．

3.4　環境独立オンライン業務プログラム処理方式

3.4.1　対話環境と OLTP 環境における業務プログラムの基本構造の相違

典型的なオンライン業務である問合せ応答処理を例にとって，対話環境とOLTP環境における業務プログラムの基本構造の相違を説明する（**図3.9**）.

(1)　対話環境

業務プログラムの起動は端末からのコマンドあるいはメニュー画面で行う.

起動されたプログラムは１つの空間に常駐する．プログラムから業務画面を出力し，端末からの入力を待ち，入力されたらDB入力など何らかの処理をし，端末の画面に結果を表示する．それが可（OK）の場合はその旨の応答を受けて，必要ならばDBを更新する．結果が否（NG）の場合は，再度処理をやり直す.

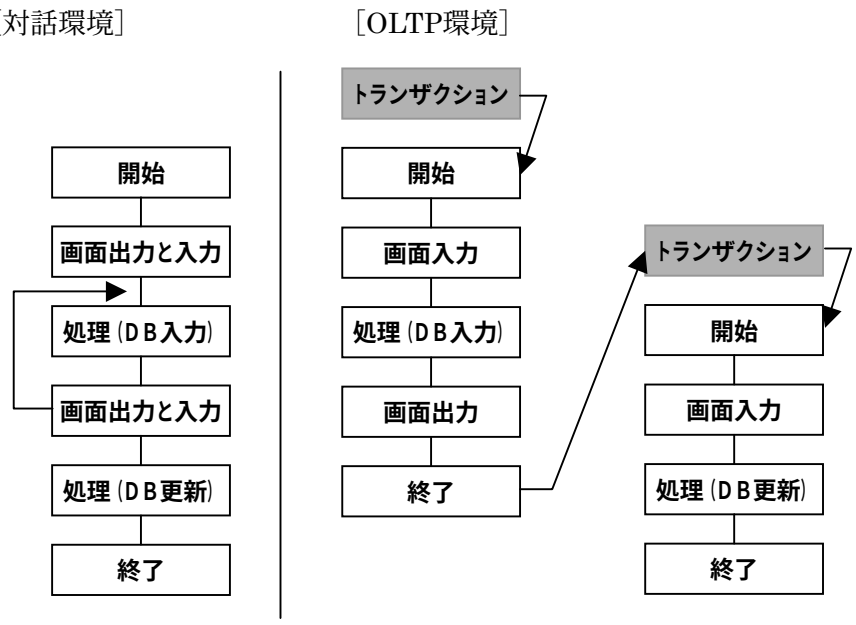


図3.9　業務プログラムの基本構造の相違　対話環境ではプログラムは常駐しているが，OLTP環境で常駐するのは入力から出力までの間だけである.

この対話環境の業務プログラムの基本構造は“画面を出力し入力⇒処理⇒画

面を出力し入力⇒・・・”という形態である．

⑵　OLTP環境

端末から開始コマンドをOLTPが受けると，定められた初期画面を表示する．このあと，この画面に端末側が応答し必要な入力を行う．これをトランザクションとしてOLTPが受け取り，キューに入れ，スケジューリングし，空間が空いたらそのトランザクションに関係づけられた業務プログラムを起動する．業務プログラムはトランザクションをメッセージとして入力（receive）しDB入力など必要な処理をし，応答メッセージ（画面）を出力（send）し，プログラムは終了する．このプログラムが一時的に占有していた空間は，OLTPが別の端末対応の業務プログラムに割り当てる．出力メッセージは出力キューを経由して端末に送られる．

すなわち，OLTP環境の業務プログラムの基本構造は“画面入力⇒処理⇒画面出力”である．

問合せ応答業務では，出力した画面に端末が応答すると，それがトランザクションとなり，それに関係づけられたプログラムがスケジューリングされて起動される．そのプログラムはDB更新などの処理を行う．2つの業務プログラム間の共有情報受け渡しは，OLTPが仲介する．

3.4.2　環境独立な処理方式

VOSKのEAGLE/4GLでは，バッチ2パターン，対話１パターンの3パターンだったが，UNIX版EAGLE/4GLでは，問合せ応答パターンを追加した．このパターンはOLTP環境と対話環境（TSS環境も含む）の両方に適用できる．この処理構造は**図3.10**に示す．

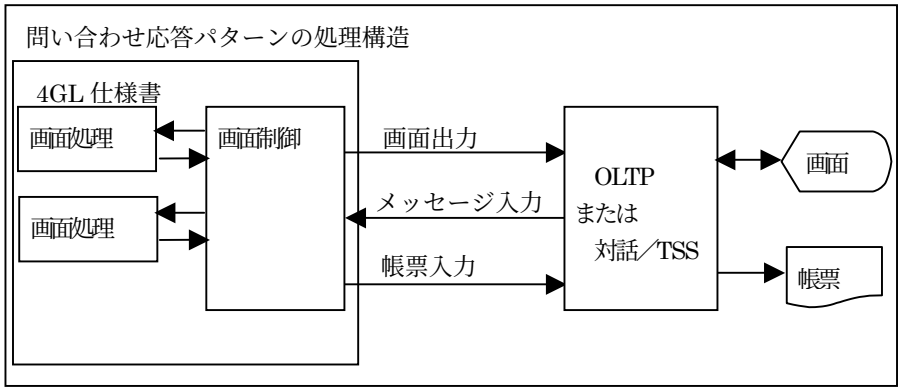


図3.10　問合せ応答パターンの処理構造　　画面制御と個々の画面処理で構成されている.

1つのオンライン業務は複数の画面で構成されることが多い. OLTP 環境のオンライン業務プログラムの作り方は，業務プログラムの基本構造が“画面入力⇒処理⇒画面出力”となっていることに起因し，複数画面の制御と個々の画面処理が渾然一体で記述されていた．これに対し，EAGLE/4GL の問合せ応答パターンでは，①画面制御と画面処理は分離して記述し，②複数のプログラムに分離されていた同じ画面の出力処理と入力処理を1つにまとめて記述するようにした.

画面制御については論理的な関係と環境（OLTPあるいは対話／TSS処理など）を指定するようにし，環境の相違はEAGLE/4GLが生成す画面制御プログラムで吸収するようにした．これにより，個々の画面に対する個別処理のみをEAGLE/4GLでは記述すればよい．問合せ応答パターンの処理概要図を**図3.11**に示す．これに対応するEAGLE/4GLの仕様書を**図3.12**と**図3.13**に示す.

EAGLE/4GLは画面制御と個々の画面処理を別々のCOBOLプログラムとして生成する.

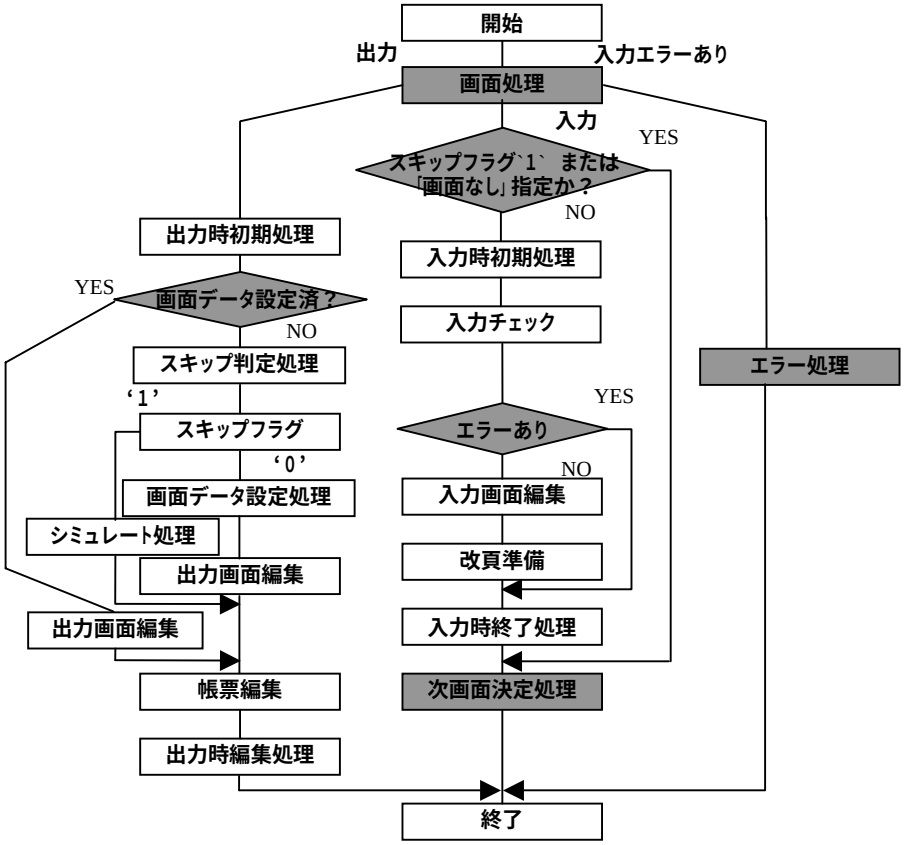


図3.11 問合せ応答パターンの処理概要　影のついた部分が画面制御部分であり，白地の部分が画面処理あるいは帳票処理である．

```
*****************************************
処理仕様書                    ＜処理仕様書名＞
*****************************************
* コメント　＜コメント＞
  パターン              問合せ応答
    [使用する画面，ファイル，及び帳票の指定]
    [起動するトランザクションの指定]
    [DAM ファイル，TAM テーブル，
    一時記憶データ受け渡し領域の指定]
  画面遷移定義
    [画面の遷移情報の定義]
  画面制御
    [PF キーの定義]
*****************************************
データ定義
*****************************************
  AP 間連絡領域
  [AP 間通信メッセージを受け取る領域の定義]
*****************************************
画面処理－Sn＜画面処理名＞
[画面処理ｵﾌﾟｼｮﾝの指定]
*****************************************
*****************************************
データ定義
*****************************************
  通信連絡領域
    [画面処理間での連絡用作業領域の定義]
  作業領域
    [処理で使用する作業領域の定義]
  遷移条件定義
[遷移条件名に対する遷移条件の定義]
```

```
*****************************************
手続き
*****************************************
  出力時初期処理
    [画面表示処理の最初に行う処理の記述]
  出力時終了処理
    [画面表示処理の最後に行う処理の記述]
  スキップ判定処理
    [画面の表示，スキップの判定処理の記述]
  画面データ設定処理
    [改頁構造に該当する画面処理の中で最初の
    画面出力時一度だけ行う（全ページ設定処
    理）の記述]
  出力画面編集
    [表示する画面の編集処理の記述]
  シミュレート処理
    [画面を表示しない時の記述]
  帳票編集
    [帳票に出力するデータの編集処理の記述]
  入力時初期処理
    [画面データの入力直後に行う処理の記述]
  入力時終了処理
    [画面入力処理の最後に行う処理の記述]
  入力チェック
    [入力データのチェック処理の記述]
  入力データ編集
    [入力データに対する加工処理の記述]
  改頁準備
    [改頁に対する加工処理の記述]
  サブ手続き　＜サブ手続き名＞
    [画面処理－Sn での共通処理の記述]
```

図3.12　　問合せ応答パターンの処理仕様書　手続きの［　］の中の処理を記述していけば処理仕様書が完成する．

3.4.3 環境独立オンライン業務プログラム処理方式のオープン環境への適用

UNIX 環境と PC 環境（Windows）に OLTP，XMAP，COBOL，DB のミドルウェアがサポートされたので，基本的な部分はほとんど変えずに EAGLE/4GL の問合せ応答パターンはこれらの環境に適用可能となった．ただし, XMAP 自身は GUI をサポート済みだが, EAGLE/4GL の画面定義では CUI しかサポートしていないので，実際の適用はメインフレームからのストレート移行など限られている．

3.5 問題点と今後の課題

3.5.1 問題点

UNIX 版 EAGLE/4GL の開発を決め 1990 年頃は EAGLE/4GL が稼動する充分なメモリ量や GUI 機能を持つオープン OS は UNIX しかなかった. しかし, その後の Windows の進歩と搭載 PC の能力向上により, 開発環境は PC が当たり前になり，UNIX 版 EAGLE/4GL は普及しなかった．PC への移植も検討したが，GUI 機能の相違が大きく開発量が膨大で，それはあきらめた．Windows と PC の進歩の予測誤りと UNIX 普及への過度の期待が敗因である．

3.5.2 今後の課題

画面制御と画面定義を分離した環境独立処理方式は，もともとCOBOL用にOLTP環境とTSS環境に適用したのがはじまりで[大野治98], EAGLE/4GLだけでなく他の言語にも適用可能である．この方式はJavaでも実験済みで，オープン環境でも適用可能ことがわかっている[鈴木文音99] ，[藤原2001]，[湯浦2002]．現在はEJB(Enterprise Java Beans)環境での適用を検討中であり，それ用の支援ツールの開発が課題である．

3.6 まとめ

本章では，第4世代言語EAGLE/4GLについて述べ，そこで開発された複数OSのOLTP環境と対話（TSS）環境に独立なオンライン業務プログラム処理方式について論じた．この方式は他の言語やオープン環境にも適用できる．現在はEJB環境での適用を検討中で，支援ツールのサポートが課題である．

第 4 章　日本語プログラム言語“まほろば”の文法と評価

4.1　はじめに

本格的な日本語プログラム言語を設計するとどのようなものになるか，これが本章の主題である

プログラム言語の中に日本語を取り入れる研究と開発は 1960 年代から始まっており，1980 年代に日本語処理システムが実用段階に至って本格化した．最初は漢字データ処理が中心テーマであったが, ソースプログラムでも日本語を利用したいとのニーズから，現在では COBOL，SQL，Prolog，ISLISP，C++の標準言語で利用者定義の識別名として日本語（漢字）の指定が可能となっている[今城 99d].

1980 年代後半には，さらに日本語化が進み，予約語も含めて日本語にした本格的な日本語プログラム言語が実用化された．ただ，日本語といっても分かち書きを前提としていた[機シ振協 88]，[玉井 90]，[今城 99a]．その理由は，非分かち書きだとコンパイラの字句解析処理で語を認識するのが格段と難しくなるからである．

その後日本語プログラム言語の研究・開発は一時下火になったが, 近頃はいくつかのチームにより本格的な日本語プログラム言語の研究が進んでいる[プロシン 99]，[宮脇 94]，[宮脇 99]，[中鉢 97]，[鈴木弘 99]，[加藤木 99]，[畠山 2000]，[兼宗 2001b]．この中で，分かち書きをしないものは宮脇と中鉢の研究であるが，中鉢の研究はまだ完成していない．宮脇の日本語 C++は C++を忠実に日本語化したものであり, 専用エディタを用いてコンパイラの字句解析での語の認識の難しさを回避している．プログラマは日本語 C++専用エディタでソースプログラムをエディットするが, そのとき語の切れ目を指定する．専用エディタはその切れ目情報をソースファイルに反映し，日本語 C++コンパイラは字句解析時にその情報を参照しながら語の切り出しを行う．

筆者が研究した本格的な日本語プログラム言語“まほろば”は日本語 C++とは異なり新規の言語として設計した．分かち書きをしない言語であるが，専用エディタは不要であり, 普通のテキストファイルのソースプログラムをコンパイラが入力して, 字句解析が可能である．このため, 語の切り出しには，“字

句・構文同時逆向き解析方式 ” という構文解析手法を考案した．まほろばの研究途中の内容については，情報処理学会プログラム・シンポジウム，プログラミング研究会およびソフトウェア工学研究会で報告し[今城 99b], [今城 2001a], 大学用教科書でコンパイラを学ぶためのミニコンパイラはまほろばで記述した[今城 2000]．本章では，研究会などでの討論やコメントを反映した “ まほろば ” の最新版の文法とその実用性について報告する．

本章の構成は次のとおりである．**4.2** では言語仕様の設計方針を述べ，4.3 にプログラム例を示す．**4.3**～**4.9** で文法を詳細に定義する．**4.10** で “ 字句・構文同時逆向き解析方式 ” について説明し，**4.11** で実用性があるかどうか評価する．**4.12** で本章のまとめを述べる．

4.2 言語仕様の設計方針

本格的な日本語プログラム言語 “ まほろば ” を新たに設計する．どのような言語仕様にするか．大きく分けると，次のいずれかの選択となる．

(1) 既存プログラム言語の完全日本語訳
　(a) C／ C++　(b) Java　(c) COBOL など
(2) 新しい言語仕様
　(a) 簡易言語／第４世代言語など　(b) 設計仕様言語
　(c) 手続き記述の新言語（第 3 世代言語）

(1)については, C++については宮脇の研究がすでにあり, 実用的ではあるが, 新たな研究としては新規性に欠ける．(2) (a)の第 4 世代言語については，分かち書きではあるがすでに筆者は開発済みであり，本論文の **3.**で報告した．(2)（b）の分野は，UML などによる図式化表現，自然語表現からの仕様抽出，システム分析・設計開発環境とプログラミング環境の一体化などが現在の主要テーマで，単なる日本語化だけでは研究テーマとしてはアピール度が低い．ここでは，まだしかるべき研究・実用化例のない（2）(c)の “ 手続き記述の新言語 ” として，日本語プログラム言語を設計することにした．

分かち書きは小学校低学年の教育現場で使用されているだけで，それ以降は分かち書きを使用しない生活をしている．さらに，分かち書きの文法の基本は文節単位であるが，文節単位にとぎれとぎれに日本語を書くのは不慣れのため意外と大変である．日本語プログラム言語“まほろば”では，“通常我々が読み書きしているように分かち書きをしない表現方法を採用し，日本語としてできるだけ不自然さがない言語仕様にする”ことを基本とする．さらに，専用エディタなどは使わず，テキストファイルのソースプログラムをコンパイラが入力して解析可能な文法であることも要件とする．

この基本方針と要件を含め，“まほろば”の言語仕様の設計方針は次の四つとした．

(1) 日本語として違和感のない仕様とする．当然，分かち書きなしの仕様である．こうしたことによる特別な制約（専用エディタ使用など）は設けない．
(2) 今後のこの分野の研究や商用化の基礎となる仕様とする．そのための技術目標を，“データ構造とアルゴリズムが記述できること”とし，PASCAL，C，COBOL，Java を参考にしながら，プログラム言語としての基本機能を用意する．
(3) 実用化を目指す場合には，市場が最も大きい事務処理分野のプログラムが記述できなければならない．そのため，COBOL でよく使われる機能はサポートする．
(4) プログラムの品質に悪影響を与える機能は，極力サポートを控える．GOTO 文やポインタ変数は少なくとも当初は用意しない．将来追加する場合でも，オプションとし，一般には使えないようにする．逆に，プログラムの信頼性や保守性を向上させる機能は積極的に支援する．たとえば，オブジェクト指向機能であるが，これは次期の課題とした．

4.3 プログラム例

まほろばの文法を説明する前に，概要理解のために例を**図 4.1** と**図 4.2** に示す．**図 4.1** は，二つの数の最大公約数を求める関数のまほろばと C 言語での記述例である．**図 4.2** は，まほろばの記述実験（**4.11** 参照）用に作成したミニコ

ンパイラのリストの一部である．このリストには，かぎかっこがネストしたとき対応が分かりやすくするため，内側のかっこに「…1「…2「…」2…」1…」のようにネスト番号が付加されており、注釈はイタリックで陰を付けて印字されている。なお，かぎかっこの使用とネスト番号の付与は日本語 C++で行っていることを参考にした．

まほろば	C 言語

［関数］最大公約数（正数１，
正数２：整数），
もどり値は整数．
［名前］大，小，余り：整数．
［処理］
正数１≧正数２のときは，
「大←正数１．小←正数２．」
それ以外のときは，
「小←正数１．大←正数２．」
 	
小=0 になるまで，次を繰り返す．
「余り←除算余り（大，小）．
大←小．小←余り．」
もどる（大）．
［終了］
	int gcd (int x, int y);　{
	
	int　b, s, r;
	if (x >= y) {
	b = x; s = y;
	}
	else{
	s = x; b = y;
	}
	while　（s != 0）（
	r = b%s;
	b = s; s = r
	};
	return(b);
	}

図 4.1　まほろばプログラム例：最大公約数（C 言語との対比）

［名前］　　*{共通領域}　{“文字”の形式}*
文字：レコード型,「位置：文字位置型．種別：文字種別型．値：文字.」
文字種別型：データ型,「ソースの終わり，空白文字，数字，英仮名漢字，
ハイフン，長音，記号，不当文字」.
文字位置型：データ型,「行，文字位置：整数」.
文字コード種別表［６５５３６］：常駐，文字種別型，初期値は不当文字.
成功失敗型：データ型,「失敗．成功.」
{字句解析処理：ソース入力，文字取得，および字の類別}
［関数］字取得（），再帰可能.
［名前］
空白：定数,“　”.
番号：整数.
呼出し回数：常駐，列挙型，「なし，１回以上」，初期値はなし.
{入力するソース関連の領域}
入力行［１０００］：常駐，文字．入力行数：常駐．整数，初期値は０.
入力文字数：常駐，整数，初期値は０.　入力文字位置：常駐，整数，初期値は０.
ソースファイル名［１００］：常駐，文字，初期値は“まほろばソース”.

```
［処理］　　{最初だけ文字コード種別表を初期化し，ソースファイル
　　をオープンする.}
 呼出し回数がなしのときは,
　「１回以上→呼出し回数．文字コード種別表初期化（）.
　　ファイルオープン（ソースファイル名）が失敗のときは,
　　 1「エラー（“ ソースファイルが開けなかった．／”）.
　　　文字・位置・行←０．文字・位置・文字位置←０.
　　　文字・種別←ソースの終わり．文字・値←空白.
　　　もどる.」1」
　　　　{ソースの入力}
 入力文字位置が０のときは,
　「ファイル入力（ソースファイル名，入力行［]，入力文字数）が失敗のときは,
　　 1「文字・位置・行←入力行番号＋１．文字・位置・文字位置←０.
　　　文字・種別←ソースの終わり．文字・値←空白.
　　　ファイルクローズ（ソースファイル名）．もどる.」1
　　入力行番号に１を足す．入力文字位置←１.
　　　　{行の先頭からの空白を無視するための処理．空白行のときは,
　　　次の行を読む．このときは，変数“文字”に値が設定されている
　　　ので，直ちにもどる．}
　　番号が１から入力文字数まで，次を繰り返す.
　　 1「入力行［番号］が空白でないときは，ぬける.
　　　入力文字位置に１を足す.
　　　番号が入力文字数のときは,
　　　　 2「０→入力文字位置．字取得（）．もどる.」2」1
　　　　{行の最後からの空白を無視するための処理}
　　番号が入力文字数から入力文字位置まで－１ずつ変化させて，次を繰り返す.
　　 1「入力行［番号］が空白でないときは，ぬける.
　　　入力文字数から１を引く.」1」
　　　　{文字に値を設定}
 入力行番号→文字・位置・行.
 入力文字位置→文字・位置・文字位置．入力行［入力文字位置］→文字・値.
 文字コード種別表［文字整数変換（文字・値)］→文字・種別.
　　　　{行の終わりのときは，次回にソース入力するための準備として
　　入力文字位置に０をセットする.}
 入力文字数が入力文字位置より大きいときは，入力文字位置に１を足す.
 それ以外は，入力文字位置←０.
 もどる.
［終了］
```

図 4.2　まほろばプログラム例：ミニコンパイラの字句解析

4.4 文法の記述方法

まほろばの文法を，次のメタ記号を用いた拡張バッカス記法で定義する．

→ ： 左辺の構文要素は，右側の構文要素で定義される．
｜ :「または」を意味する．
＜ ＞ ： 構文要素の非終端記号を囲む．
終端記号は，＜ ＞で囲まない．
｛ ｝:｛ ｝ の中の要素を０個以上並べたもの．
［ ］:［ ］の中の要素を０または１個書いたもの．終端記号の中で使う［ ］，｛ ｝，および＜＞は，メタ記号と区別するために，**【 】**，**｛ ｝**，および**＜＞**で表現する．
… ：連続する自明な文字の省略を意味する．

4.5 符号系と文字の種類

まほろばでは漢字符号系の利用を前提にしている．漢字符号系は多くの文字や記号を含むので, 豊富な表現が可能となるが, それだけに文法も複雑になる．

4.5.1 符号系

まほろばのソースプログラムは，次の二つの符号系で記述する．

(1) 英数字符号系：英字や数字の記述が可能な符号系
(2) 漢字符号系：漢字記述が可能なマルチオクテット文字の符号系

具体的な符号系，シフトコード（符号系切り替えの制御文字）の有無，シフトコードの値，およびソースプログラム先頭および各行のシフト状態は，コンパイラ作成者（implementer）が定める． 以下では符号のことを，誤解のないかぎり“ 文字 ”という．

例：英数字符号系は，JIS X 0201 の 8 ビット符号系．漢字符号系は，JIS X 0208 のシフト符号化表現（シフト JIS）の漢字符号系．シフトコードなし．

4.5.2　計算機文字集合とまほろば文字集合

＜計算機文字集合の文字＞→＜計算機で使える文字＞
＜まほろば文字集合の文字＞→ ＜英数字符号系まほろば文字集合の文字＞
　　｜＜漢字符号系まほろば文字集合の文字＞

(1)　まほろばソースプログラムの文字定数と注釈の中では，それが稼動する計算機で使用可能な符号系のすべての文字が使用できる．これを“計算機文字集合”という．計算機文字集合はコンパイラ作成者が定める．ただし，文字定数の最後を示す右引用符と，注釈の最後を示す右中括弧の使用については制約があり，それについては **4.6** で述べる．
(2)　まほろばソースプログラムの文字定数と注釈の中以外で使える文字に制約があり，そこで使用できる文字を“まほろば文字集合”という．

まほろば文字集合を構成する英数字符号系文字集合と漢字符号系文字集合の文字については，**4.5.3** と **4.5.4** に示す．続いて，名前などを構成する文字，句読点など区切りとなる記号，引用符と括弧，演算などに用いる記号の 4 種類に分けて **4.5.5**～**4.5.8** で説明する．

4.5.3　英数字符号系まほろば文字集合

＜英数字符号系まほろば文字集合の文字＞→＜空白＞｜＜数字＞｜＜英字＞
　　｜＜アンダスコア＞｜＜英数字符号系の記号＞
＜英数字符号系の記号＞→ ＜ピリオド＞｜＜コンマ＞｜＜小数点＞
　　｜＜コロン＞｜＜中点＞｜＜アポストロフィ＞
　　｜＜コーテーションマーク＞｜＜小括弧＞｜＜中括弧＞｜＜大括弧＞
　　｜＜符号＞｜＜英数字符号系の演算記号要素＞
＜英数字符号系の演算記号要素＞→＋｜－｜＊｜／｜＝｜＞｜＜

(1) JIS X 0201 には片仮名（いわゆる半角片仮名）が含まれているが，これは英数字符号系まほろば文字集合には含めない．

(2) 英数字符号系文字集合の文字で，文字定数と注釈の中以外のものは，翻訳時のソースプログラムの読み込み直後に対応する“漢字符号系文字集合”の文字に変換される．その後に構文が調べられる．この規則により，英数字符号系（半角）の名前「ABC」と漢字符号系（全角）の名前「ＡＢＣ」は，等価となる．なお，英小文字の名前「abc」と英大文字の名前「ABC」は，等価にはならない．

(3) 文字定数を囲む引用符（アポストロフィとコーテーションマーク）に対応する文字は JIS X 0208 には含まれていない．これらが漢字符号系になければ，次のように変換される．

変換前	変換後
左側のアポストロフィ	左シングル引用符
右側のアポストロフィ	右シングル引用符
左側のコーテーションマーク	左ダブル引用符
右側のコーテーションマーク	右ダブル引用符

4.5.4 漢字符号系まほろば文字集合

＜漢字符号系まほろば文字集合の文字＞→＜空白＞｜＜数字＞｜＜英字＞
　｜＜平仮名＞｜＜片仮名＞｜＜漢字＞｜＜アンダスコア＞
　｜＜準仮名漢字＞｜＜漢字符号系の記号＞
＜漢字符号系の記号＞→ ＜句読点＞｜＜小数点＞｜＜コロン＞｜＜中点＞
　｜＜引用符＞｜＜括弧＞｜＜波ダッシュ＞｜＜符号＞
　｜＜漢字符号系の演算記号要素＞
＜漢字符号系の演算記号要素＞→
　＋｜－｜×｜÷｜＝｜≠｜＞｜＜｜≧｜≦｜←｜→

漢字符号系の記号は，英数字符号系のそれと比べ以下の相違がある．

(1) 漢字と仮名が使える．

⑵ “ 々 ” など，仮名や漢字の繰返し記号が使える．
⑶ 句読点として，ピリオドとコンマ以外に，句点“ 。”と読点“ 、”が使える．
⑷ 左右の区別のないアポストロフィとコーテーションマーク以外に，左右の区別があるシングルとダブルの引用符が使える．なお，左右の区別のないアポストロフィとコーテーションマークは，漢字符号系に含まれなくてもよい．
⑸ 左かぎ括弧の “「 ” と，右かぎ括弧の “ 」” が使える．
⑹ 演算記号の書き方が異なる．これは，英数字符号系では，本来の算術記号（×，÷，≠，≧，≦）や矢印（←，→）が使えないため，代替表現を用いていることに起因する．

4.5.5 名前などを構成する文字

＜数字＞→０｜１｜２｜３｜４｜５｜６｜７｜８｜９
＜英字＞→＜英小文字＞｜＜英大文字＞
＜英小文字＞→ａ｜ｂ｜ｃ｜ｄ｜…｜ｙ｜ｚ
＜英大文字＞→Ａ｜Ｂ｜Ｃ｜Ｄ｜…｜Ｙ｜Ｚ
＜平仮名＞→あ｜い｜…｜ん
＜片仮名＞→ア｜イ｜…｜ン
＜漢字＞→＜漢字符号系の漢字＞
＜英単語連結文字＞→＜アンダスコア＞
＜アンダスコア＞→＿
＜準仮名漢字＞→＜長音記号＞｜＜仮名漢字繰返し記号＞｜＜漢数字ゼロ＞
＜長音記号＞→ー
＜仮名漢字繰返し記号＞→ゝ｜ゞ｜ヽ｜ヾ｜々
＜漢数字ゼロ＞→〇

⑴ ＜平仮名＞と＜片仮名＞には，「ゃ」などの小文字や，「バ」，「ポ」などの濁音，半濁音を含む．単独の濁点や半濁点は含まない．
⑵ ＜漢字＞には，記号（JIS X 0208 のけい線記号，単位記号，多くの学術記号など）は含まない．たとえば，“ ＃記号 ” や “ @地名 ” は，まほろば

の名前では使えない.

(3) JIS X 0208 に存在する漢字，キリール文字およびギリシャ文字は，まほろばでは＜漢字＞に含める．JIS X 0208 以外の漢字符号系を使用するときは，＜漢字＞の文字に何を含めるかはコンパイラ作成者が定める.

4.5.6 句読点など区切りとなる記号

＜空白＞→△
＜句読点＞→＜句点＞｜＜読点＞｜＜ピリオド＞｜＜コンマ＞
＜句点＞→。
＜読点＞→、
＜ピリオド＞→.
＜コンマ＞→,
＜小数点＞→.
＜コロン＞→：
＜中点＞→・
＜波ダッシュ＞→〜

本論文で，空白を特に意識して表現するときは，△で表現する.

4.5.7 引用符と括弧

＜引用符＞→＜左右シングル引用符＞｜＜左右ダブル引用符＞
　｜＜アポストロフィ＞｜＜コーテーションマーク＞
＜括弧＞→＜小括弧＞｜＜中括弧＞｜＜大括弧＞｜＜かぎ括弧＞
＜左右シングル引用符＞→‘｜’
＜左右ダブル引用符＞→“｜”
＜アポストロフィ＞→’
＜コーテーションマーク＞→”
＜小括弧＞→（｜）
＜中括弧＞→｛｜｝

＜大括弧＞→【｜】
＜かぎ括弧＞→「｜」

4.5.8 演算などに用いる記号

＜英数字符号系の演算記号要素＞→＋｜－｜＊｜／｜＝｜＞｜＜
＜漢字符号系の演算記号要素＞→
　　＋｜－｜×｜÷｜＝｜≠｜＞｜＜｜≧｜≦｜←｜→
＜正負符号＞→＋｜－

(1) 漢字符号系の演算記号の中で，英数字符号系の演算記号に含まれない次の文字は，英数字符号系では 2 文字で表現される．これらは，次のように変換される．ここの変換前の書き方は，漢字符号系の中では許されない．

変換前	変換後	変換前	変換後
＜－	←	－＞	→
＊	×	／	÷
＞＝	≧	＜＝	≦

4.6 語句の種類：名前，定数，予約語など

まほろばソースプログラムは，語句（トークン）の連続で構成される．語句は，一つ以上の文字が連続した文字列で構成される．語句の種類は，利用者が定義する名前, 利用者が定義する定数, 予約語，１文字または２文字の記号列，一つ以上の空白が連続した空白列，または注釈のいずれかである．以下では，語句のことを，誤解のないかぎり，語ということが多い．

＜語句＞→＜名前＞｜＜定数＞｜＜予約語＞｜＜記号列＞｜＜空白列＞
　　｜＜注釈＞
＜名前＞→＜先頭文字＞ {[＜後続文字＞］＜最終文字＞}
＜先頭文字＞→＜英字＞｜＜平仮名＞｜＜片仮名＞｜＜漢字＞
　　｜＜漢数字ゼロ＞

＜後続文字＞→＜先頭文字＞｜＜数字＞｜＜英単語連結文字＞
　　｜＜準仮名漢字＞
＜最終文字＞→＜先頭文字＞｜＜数字＞｜＜準仮名漢字＞

＜定数＞→＜数値定数＞｜＜文字定数＞
＜数値定数＞→＜整数＞｜＜固定小数点数＞｜＜浮動小数点数＞
＜整数＞→［＜正負符号＞］＜数字＞｛＜数字＞｝
＜固定小数点数＞→
　　［［＜正負符号＞］＜整数＞］＜小数点＞＜数字＞｛＜数字＞｝
＜浮動小数点数＞→＜仮数部＞＜指数部＞
＜仮数部＞→＜整数＞｜＜固定小数点数＞
＜指数部＞→Ｅ＜整数＞
＜文字定数＞→
　　＜左引用符＞＜計算機文字集合＞｛＜計算機文字集合＞｝＜右引用符＞
　　｜＜文字定数＞つなぐ＜文字定数＞
＜左引用符＞→＜左シングル引用符＞｜＜左ダブル引用符＞
　　｜＜アポストロフィ＞｜＜コーテーションマーク＞
　＜右引用符＞→＜右シングル引用符＞｜＜右ダブル引用符＞
　　｜＜アポストロフィ＞｜＜コーテーションマーク＞

＜記号列＞→ ＜句読点＞｜＜コロン＞｜＜中点＞｜＜小括弧＞｜＜大括弧＞
　　｜＜波ダッシュ＞｜＜漢字符号系の演算記号要素＞
　　｜＜英数字符号系の演算記号要素＞｜＜英数字符号系２文字演算記号＞
＜英数字符号系２文字演算記号＞→－＞｜＜－｜＞＝｜＜＝
＜空白列＞→＜空白＞｛＜空白＞｝
＜注釈＞→｛［＜計算機文字集合の文字＞］｝

例： 名前：　公約数，キューリー夫人，佐々木，J_Kenedy
　　整数：　１２３，　－５８
　　固定小数点：　１２３．４５　　－５７．９９９
　　浮動小数点：　６．７Ｅ５０　　－０．５Ｅ－５

５３２Ｅ４５

文字定数：　“日本語”　‘まほろば’　“ABC”　‘abc’

“シングル引用符の「‘」と「’」で囲む”

文字定数の連結：　“ＡＢＣ”つなぐ“ＤＥＦ”

(1) 固定小数点数と浮動小数点数は小数点を含む. 小数点とピリオドは符号系の中で同じ符号である．これらを区別するため，ピリオドの前後が数字の場合は，それは小数点とみなす.

(2) 左引用符と右引用符は，同種の引用符でなければならない．左引用符に対応する右引用符は，文字定数に含むことができない．右引用符を定数の中に書くときは，異なった種類の引用符でその定数を囲めばよい．文字定数の間に「つなぐ」と書くと，定数を連結できる.

(3) 通常のプログラム言語の予約語は名前として使用できないが，まほろばの予約語は名前として使用できる．予約語と同じ名前が，予約語と認識されるか，名前と認識されるかは，構文に依存する.

(4) 正負符号，小数点，および引用符は定数を構成し，中括弧は注釈を構成するので，語句の分類では，記号列に含めていない.

(5) 注釈の中には右中かっこ“}”は記述できない.

(6) 注釈は，まほろばコンパイラの語句解析後は，あたかも一つの空白がそこにあるかのように，扱われる．注釈の中に右大括弧があっても，それは注釈の終わりとみなされる.

(7) それぞれの文字数の上限は，次のとおりとする.

  (a) 名前と予約語：30 文字

  (b) 数値定数： 30 文字（小括弧を除く）

  (c) 単独の文字定数：120 文字（左右の引用符を除く）

4.7 行と空白の規則

空白の規則は日本語では英語と異なる．日本語プログラム言語であるまほろばの空白の規則は，英語を基調とした標準プログラム言語とは異なる.

(1) ソースプログラムは，１行以上の行で構成される．行の終わりが，物理的な最後の場合か，“ 行の終わりを示す特別な文字 ” が現れた場合かのいずれであるかは，コンパイラ作成者が定める．

(2) 日本語には，「前の行の最終文字が，後ろの行の先頭文字に連続する」という性質がある．たとえば，次の 2 行があったとき

「佐藤さんは東△」

「△京駅に着いた.」

空白△の存在は無視され，次のように解釈される。

「佐藤さんは東京駅に着いた.」

これと同じように, まほろばソースプログラム上の, “ 行の終わりを示す特別な文字 “ ，行の左端部分の空白列，および行の右端部分の空白列は，まほろばコンパイラでは，あたかも存在しないかのように扱われる．

(3) 行が全部空白で構成される空白行も，まほろばコンパイラでは，あたかも存在しないかのように扱われる．

(4) （3）と(4)の規則は，文字定数が行末で終わっていない場合にも適用される．

(5) 英語では空白は語と語を区切る．日本語でも，行の途中に現れる空白は，分かち書きという文法が適用され，語と語，あるいは文節と文節を区切るために用いられる．まほろばコンパイラも，行の途中の空白列に限り，語と語の区切り文字として処理する．行の途中の空白列とは，厳密には，行の最左端および最右端の空白列，文字定数の中の空白列，および注釈の中の空白列を除いた空白列のことである．

4.8 プログラムの構成と名前の定義

4.8.1 構文規則の留意事項

煩雑さをさけるため，**4.8** と **4.9** の構文規則は次のように記述している．

(1) 利用者が定義する名前と定数（変数名，定数名，関数名，データ型名，列挙名，数値定数，文字定数）は，終端記号として扱い，メタ記号の＜ ＞では，囲まない．

(2) 句点と読点は，それぞれ 2 種類存在するが，構文規則中では“．”と“，”で代表する．
(3) コンパイラの語句の解析処理は，ソースプログラムの読み込みより後で行うので，語句解析処理段階では，英数字符号系から漢字符号系への変換は終わっていると考えてよい．よって，漢字符号系の字や語だけを対象に構文規則を記述する．

4.8.2 プログラムの構成

まほろばのプログラムは１個以上の関数で構成する．関数は関数先頭部，名前部，処理部の三つの部から成り，関数の間で共用する名前はプログラムの先頭の名前部に記述する．

＜プログラム＞→［＜名前部＞］＜関数＞｛＜関数＞｝
＜関数＞→＜関数先頭部＞［＜名前部＞］＜処理部＞
＜関数先頭部＞→ 【関数】関数名（[＜仮引数＞｛，＜仮引数＞｝]）
　　［，もどり値は＜データ型＞］［，再帰可能］.
＜名前部＞→【名前】｛＜定数名定義＞｜＜変数名定義＞｜＜データ型名定義＞
　　｜＜関数の引数の型定義＞｝
＜処理部＞→【処理】｛＜文＞｝｛＜関数＞｝【終了】
＜仮引数＞→＜変数名定義＞

(1) 先頭の＜名前部＞に書いた名前は，プログラム中のすべての関数で参照できる．
(2) 関数先頭部には，関数名と仮引数を記述する．もどり値を指定した関数は，式の中で参照する．もどり値を指定しない関数は，呼出し文で参照する．関数のもどり値は，「もどる（返却値）」のように，復帰文で指定する．“再帰可能”と指定すると，その関数は再帰性をもつ．
(3) 名前部では，変数名，定数名，関数名，列挙名およびデータ型名を指定できる．
(4) 処理部では，文を 0 個以上記述する．

(5)　仮引数の変数名定義では常駐と初期値は指定できない.

4.8.3　定数名と変数名の定義

＜定数名定義＞→定数名｛，定数名｝:［広域,］定数，＜定数＞.
＜変数名定義＞→変数名【＜配列要素数＞｛，＜配列要素数＞｝】
　　｛，変数名【＜配列要素数＞｛，＜配列要素数＞｝】｝:
　　［常駐,］［広域,］＜データ型＞［，初期値は＜定数＞］.
＜配列要素数＞→数値定数｜定数名

(1)　定数名や変数名などの定義では,「商品コード：….」 のように，先ず名前を書き，次にコロン“ ： ”を続け，その後に具体的な内容を定義し，句点“ . ”で終わる.
　例：エラー：定数“ error ”.
(2)　変数名に常駐の指定があると，その変数の領域はその変数を含む関数が終了したときの値のまま保存され、その関数が再び呼ばれると同じ領域が使用される．変数名に常駐の指定がないと，関数が呼び出されるたびに値が初期化される．プログラム先頭の名前部の変数に常駐の指定がなくても，常駐が仮定される.
(3)　広域の指定がある名前は，その定義がある関数に含まれる関数からも参照可能である．なお，プログラム先頭の名前部の名前は広域の指定がなくても，プログラム中のすべての関数から参照できる.
(4)　変数は，構造を持たないものと，持つもの（構造体）に分かれる．構造体には，配列とレコード型がある.
(5)　配列は，変数名の後に要素数を書き，それを大括弧［ ］で囲むことにより定義する．要素数は，１以上の整数でなければならない．配列参照時の要素数は，１から始める.
　例：都道府県人口［４７］:整数，初期値は０.
　　配列の参照は，都道府県［県コード］などと記述する.
(6)　レコード型については，**4.8.5** で述べる.

4.8.4 データ型

まほろばには，整数型，実数型，10 進数型，数字編集型，文字型，レコード型，領域共用型，列挙型，範囲型，および利用者が定義するデータ型の 10 種類のデータ型がある．

＜データ型＞→整数［＜精度＞］｜実数［＜精度＞］
　｜１０進数＜けた数＞けた[,小数点以下＜けた数＞けた］［＜格納種別＞］
　｜数字編集＜編集指定＞ ｜［＜文字数＞］文字［＜精度＞］
　｜＜レコード型＞｜＜領域共有型＞｜＜列挙型＞｜＜範囲型＞
　｜データ型名
＜けた数＞→数値定数
＜文字数＞→数値定数
＜精度＞→（短）｜（中）｜（長）
＜格納種別＞→（アンパック）｜（パック）｜（バイナリ）

例：点数：整数３けた．　……誤りの例
　　身長：１０進数４けた，小数点１けた．……正しい例
　　氏名：１０文字．　……正しい例

(1) ポインタ型やビット（ブール型）はサポートしていない．可変長レコードもサポートしていない．
(2) 10 進数は SQL や COBOL にもある．数字編集も COBOL にある．いずれも事務処理向けの機能である．
(3) 編集指定は COBOL の PICTURE 句での数字編集の書き方を文字定数の中に記述する．
　例：金額：数字編集“￥￥￥，￥￥￥，￥￥９”
(4) けた数，文字数は，1 以上の整数を指定する．1 文字のときは，文字数を省略できる．
(5) 精度ではそれぞれの型の実行時に確保する大きさを指定する．実際の大きさは，コンパイラ作成者が定める．精度を指定しない場合は“中”が仮定

される．通常は，整数では短（2 バイト），中（4 バイト），長（8 バイト），実数では短（4 バイト），中（8 バイト），長（16 バイト），文字では短（1 バイト），中（2 バイト），長（4 バイト）である．

(6) 格納種別では 10 進数のデータの格納形態を指定する．格納種別を指定しないと，アンパックが仮定される．

4.8.5 レコード型と領域共用型

＜レコード型＞→レコード型，「＜変数名定義＞｛＜変数名定義＞｝」
＜領域共用型＞→領域共用型，「＜変数名定義＞｛＜変数名定義＞｝」

(1) レコードは，複数の変数を先頭から順に並べたものである．
(2) 領域共用型は，同一領域を異なる名前やデータ型で共有するときに用いる．
(3) レコード型や領域共用型に含まれる変数を参照するときには，その前にレコード型または領域共用型の変数名と中点“ ・ ”を書く．

例： レコード型：氏名と複数の科目で構成される成績は，次のように書く．
成績：レコード型，「氏名：１５文字．科目［１０］：整数．」．

例： 領域共用型：アメリカなど外国の住所と日本の住所を，同一領域に格納するときには，次のように書く．
住所：領域共用型，「英語：５０文字（小）．日本語：２５文字（中）．」．

例： 参照：上の例では，
“ 成績・氏名 ”，“ 成績・科目［科目番号］”，“ 住所・英語 ”，“ 住所・日本語 ”
などと参照する．

4.8.6 列挙型，範囲型，利用者定義のデータ型

＜列挙型＞→「列挙名｛，列挙名｝」
＜範囲型＞→
「＜定数参照＞［～＜定数参照＞］｛，＜定数参照＞［～＜定数参照＞］｝」
＜データ型名定義＞→データ型名：［広域］データ型，＜データ型＞．

⑴　変数の値を限定する場合に，列挙型と範囲型を用いる．
⑵　列挙型では，実行時に取る値に列挙名をつける．列挙型の値は，先頭から０．１．２…が対応し，それらが実行時に使われる．
⑶　列挙型は，整数型とみなされる．

例：　列挙型：“色の種類”という変数の値が赤，青，黄のいずれかしかない場合には，次のように書く．実行時に実際に取る値は整数で，赤は０，青は１，黄は２である．

三色：データ型,「赤，青，黄」.
色の種類：三色.

例：　列挙型：　所属：「総務，経理，技術，営業」.
これは次と同じであるが，データ型名を指定しなくてよい．

部名：データ型,「総務，経理，技術，営業」.
所属：部名.”

例：　範囲型：　５段階評価：データ型,「１～５」.
評価：５段階評価.

4.8.7　関数の引数の型指定

外部で定義している関数の引数やもどり値の型を指定する．この指定があると，関数参照との間で整合性がチェックされる．指定がないとチェックされない．標準関数については，記述不要である．

＜関数の引数の型定義＞→関数名：[広域,]関数（{＜データ型＞
{，＜データ型＞}}）[，もどり値は＜データ属性＞].

例：　最大公約数：関数（整数，整数），もどり値は整数.

4.8.8　名前の参照

＜関数参照＞→関数名（[＜実引数＞ {，＜実引数＞}]）

＜変数定数参照＞→＜変数参照＞｜＜定数参照＞
＜変数参照＞→｛＜上位名＞・｝変数名［【［＜添字＞｛，＜添字＞｝］】｝
＜定数参照＞→＜定数＞｜定数名｜列挙名
＜定数＞→数値定数｜文字定数
＜上位名＞→＜変数参照＞
＜実引数＞→＜式＞
＜添字＞→＜式＞

(1) 配列全体を参照する場合には，【　】の中の添字は書かない．
(2) 列挙名は，その列挙名を含む列挙型の変数名と対応するときだけ，参照できる．
(3) 利用者が定義する名前には，関数名，変数名，定数名，列挙名，データ型名の五つがある．それらは，同一スコープ内で，かつ関数名，（変数名，定数名，列挙名），データ型名の３つの各グループ内で一意でなければならない．
    (a) 関数名の直後には，“（”が必ず必要だが，“（”が変数名の直後にくることはありえない．これで，関数名と変数名は区別できる．
    (b) レコード型，領域共有型に含まれる変数名は，修飾により一意にできれば，同じ名前であってもよい．修飾とは，その変数名の直前に上位レベルの名前と中点“・”をつけることである．

4.9 文：処理の記述

4.9.1 文の種類

処理部には 0 個以上の文を記述する．文には代入文や二分岐文などがある．

＜文＞→＜重文＞｜＜空文＞｜＜呼出し文＞｜＜復帰文＞｜＜代入文＞
　｜＜二分岐文＞｜＜多分岐文＞｜＜繰返し文＞｜＜脱出文＞
＜重文＞→「＜文＞｛＜文＞｝」
＜空文＞→何もしない．
＜呼出し文＞→＜関数参照＞［を呼ぶ］．

＜復帰文＞→もどる［([＜式＞])].

⑴　重文は，他の言語のブロックに対応するが，局所的な変数は書けない.
⑵　復帰文で，もどり値のある関数を参照するときには，かえり値を（　）の中に書く.

4.9.2　代入文

＜代入文＞→＜変数参照＞＝＜式＞.　｜＜変数参照＞←＜式＞.
　　｜＜式＞→＜変数参照＞｜＜日本語代入文＞.
＜式＞→［＜加減演算子＞］　＜項＞｛＜加減演算子＞＜項＞｝
＜項＞→＜因子＞｛＜乗除演算子＞＜因子＞｝
＜因子＞→＜変数参照＞｜＜定数参照＞｜＜関数参照＞｜（式）
＜乗除演算子＞→×｜÷
＜加減演算子＞→＋｜－

⑴　異なる型の代入はできない．ただし，異なる数値どうしの代入は可能である．精度が異なる文字型の間の代入はできない．異なる型のデータ間の変換と除算の余りは，標準関数でサポートする．文字型は四則演算できない.
⑵　式の中を項と因子の階層に分けているのは，×と÷は＋や－より優先順序が高いことを規則として明記しなくて済むからである．式の最下位の要素は，変数名か定数である．括弧により，優先順序を変更できる.
⑶　日本語代入文については，**4.9.6** で述べる.

例：　合計点←国語＋数学＋英語.
　－ａ＋ｂ，項＋項，項－項，因子×因子，因子÷因子

4.9.3　二分岐文と条件

＜条件文＞→＜条件＞のときは，＜文＞｛それ以外は，＜文＞｝
＜条件＞→＜単純条件＞｜＜複合条件＞

＜単純条件＞→＜比較条件＞｜（＜条件＞）｜（＜条件＞）でない
＜比較条件＞→ ＜式＞＜比較演算子＞＜式＞［でない］｜＜日本語比較条件＞
＜比較演算子＞→＝｜≠｜＞｜≧｜＜｜≦
＜複合条件＞→＜積条件＞｛または＜積条件＞｝
＜積条件＞→（＜単純条件＞）｛および（＜単純条件＞）｝

⑴ 異なる型の比較はできない．ただし，数値どうしの比較は可能である．精度が異なる文字型の間の比較はできない．
⑵ 日本語比較条件については，**4.9.6** で述べる

例：点数≧６０のときは，A→Z．それ以外は，B→Z．

4.9.4 多分岐文

＜多分岐文＞→ ＜比較主体＞の値により分かれる．
「＜比較対象＞の場合，｛＜比較対象＞の場合,｝＜文＞
｛＜比較対象＞の場合，｛＜比較対象＞の場合,｝＜文＞｝
｛その他の場合，＜文＞｝」
＜比較主体＞→＜変数参照＞｜＜関数参照＞｜（式）
＜比較対象＞→＜変数定数参照＞［～＜変数定数参照＞］

⑴ 比較主体と比較対象を順番に比較し，一致した場合の対応する分岐が選択される．各分岐の終わりに至ると，制御は多分岐文の最後の“」”に移る．

例： 所属の値により分かれる．
「総務の場合，「…」
営業の場合，「…」
その他の場合，「…」」

4.9.5　繰返し文と脱出文

＜繰返し文＞→［＜繰返し条件＞,］次を繰り返す．＜重文＞
＜繰返し条件＞→［最後で条件判定,］＜条件＞になるまで
　　｜［最後で条件判定,］＜条件＞の間
　　｜＜変数参照＞が＜変数定数参照＞から＜変数定数参照＞ まで
　　［＜制約参照＞ずつ変化させて］
＜脱出文＞→ぬける．｜繰返しの先頭にもどる．

⑴　脱出文は，繰返し文の中以外には書けない．
⑵　脱出文を含む繰返し文が入れ子になっているとき，その脱出文は最も内側の繰返し文に対し有効である．
　　注：多分岐文の分岐の途中に脱出文があっても，制御は多分岐文の最後の“」”には移らない．制御が移る先は，脱出文を直接含む繰返し文に続く重文の最後の“」”である
⑶　“最後で条件判定”があると，条件判定が繰返しの最後で行われる．
⑷　１ずつ変化させるときは，“＜制約参照＞ずつ変化させて”を省略できる．

例：ａ←１．
　　次を繰り返す．
　　「ａ＞１００のときは，ぬける．ａ←ａ＋１．・・・．」
例：ファイル入力（マスタ，入力領域）が終りになるまで，次を繰り返す．
　　「配列要素が１から１０まで１ずつ変化させて，次を繰り返す．
　　「Ａ＞Ｂのときは，ぬける．・・・．」
　　・・・．」
　ここの“ぬける”は，内側の繰返し文“配列要素が…”に対し有効である．

4.9.6　日本語代入文と日本語比較条件

使用頻度の高い代入と比較条件に限って，記号を使わずに特定の日本語表現を可能としている．これにより，プログラムの大部分が普通の日本語に近い表

現で書くことができる.

＜日本語代入文＞→＜日本語単純代入文＞｜＜日本語加算文＞
　　｜＜日本語減算文＞
＜日本語単純代入文＞→＜変数定数参照＞を＜変数参照＞に入れる.
｜＜変数参照＞に＜変数定数参照＞を入れる.
＜日本語加算文＞→＜変数定数参照＞を＜変数参照＞に足す.
　　｜＜変数参照＞に＜変数定数参照＞を足す.
＜日本語減算文＞→＜変数定数参照＞を＜変数参照＞から引く.
　　｜＜変数参照＞から＜変数定数参照＞を引く.
＜日本語比較条件＞→
　　＜変数定数参照＞が＜変数定数参照＞［＜日本語比較＞］［でない］
＜日本語比較＞→＜より大きい＞｜＜より小さい＞

(1) 二分岐文や繰返し文にあらわれる表現で,“ のときは ”や“ になるまで ”などの前に“ でない ”がくる場合は，日本語としておかしな表現となってしまう．これらは禁止し，対応する語尾変化後の表現を書けるようにしている．語尾変化後の表現は，構文規則には反映していない.

でないのときは	⇒	でないときは
でないになるまで	⇒	でなくなるまで
でないの間	⇒	でない間
より大きいのときは	⇒	より大きいときは
より小さいのときは	⇒	より小さいときは
より大きいでないのときは	⇒	より大きくないときは
より小さいでないのときは	⇒	より小さくないときは

注:“ 地名が“ 東京 ”でないでない“ の二重否定は日本語として奇異である.
これは書けないように構文規則を定めてある.

例： ＡをＢに入れる.		ＡにＢを入れる.
ＡをＢに足す.		ＡにＢを足す.
１をＡに足す.		Ａに１を足す.

AからBを引く.	BをAから引く.
Aから１を引く.	１をAから引く.
AがBのときは,	AがBでないときは,
AがBより大きいときは,	AがBより大きくないときは,
AがBより小さいときは,	AがBより小さくないときは,
AがBになるまでは,	AがBでなくなるまでは,
AがBより大きくなるまでは,	AがBより大きくなくなるまでは,
AがBより小さくなるまでは,	AがBより小さくなくなるまでは,
AがBの間は,	AがBでない間は,
AがBより大きい間は,	AがBより大きくない間は,
AがBより小さい間は,	AがBより小さくない間は,

4.9.7 標準関数

次のものなど，一般によく使用する機能を標準関数でサポートする.

(1) 文字列の部分参照：開始文字位置と長さが可変となるので，ある程度のポインタ処理が可能である.
例：部分文字列（漢字列，開始文字位置，長さ）
(2) 文字列の連結と分離
(3) 剰余演算
(4) 異なるデータ型の変換
(5) ファイルの入出力（レコード単位，フィールド（項目）単位）
(6) ファイルの入出力（COBOL の順ファイル、行順ファイル（テキストファイル)、相対ファイル，索引ファイル相当機能）
(7) 数値の編集
(8) 日付や時刻の取得

4.10 字句・構文同時逆向き解析方式

本節では，**4.2** で述べたまほろばの言語仕様設計方針(1)の「日本語として違

和感のない仕様とする．当然，分かち書きなしの仕様である．こうしたことによる特別な制約（専用エディタ使用など）は設けない.」の後段の要件を達成するためにコンパイラで採用する処理方式について論じる．

4.10.1 課題と方式概要

まほろばの構文は,“ 部名が技術部のときは，1 を部コードに入れる．”というような日本語構文に特長がある．まほろばの日本語構文は，C や COBOL など英語を基調としたプログラム言語と異なり，文末に予約語の動詞があり，語が空白などで区切られていない．このため，既存のコンパイラ語の切り出し処理は使用できない．

文節を直感的に定義すると,“ AがBのときは，X（C）を呼ぶ. ”というような構文において句読点で区切られた単位のことである．

英語を基調としたCOBOLなどのプログラム言語では動詞などの予約語が文（節）の先頭にくるが，まほろばは日本語を基調にしているので動詞などの予約語が文（節）の最後にくるということに着目し，文（節）を逆方向に調べていけば，構文が解析できるのではないかと考えた．具体的には，まほろばコンパイラの構文解析方式として,「構文解析をある程度やりながら,字句解析を行う」という字句解析と構文解析を一体化させた“ 字句・構文同時逆向き解析方式 ”を考えた．これは,「語分解をせずに，先ず句読点まで文（節）をスタックし，スタック末尾の動詞などの予約語の種類を調べる．次に通常の字句解析とは逆向きに，右端から左方向に助詞を探し，助詞の左右が変数として定義されているか調べる．定義されていないと，さらに左を探す.」という方式である．関数部や名前部にも日本語構文は現れるが，複雑なのは処理部なので，本節では，処理部で採用したアルゴリズムを説明する．

4.10.2 文節の終了を示す区切り文字

ここでは，文節をどのように認識するかについて考察する．

本節では,“ 、”と“ , ”，および“ 。”と“ . ”を，まとめて読点および句点ということにする．

たとえば,

所属が技術部のときは，特別処理（社員コード，資格）を呼ぶ.

という文は，二つの文節で構成される．この中で最初の読点（,）は文節を区切るが，社員コードの直後の読点（,）は文節を区切らない．構文解析の関心の対象は文節を区切る句読点やかぎ括弧であり，それは括弧や引用符に囲まれていないものである．これを“レベル１句読点”,“レベル１かぎ括弧”という．本節では，誤解のない限り，レベル１を省略して，単に句読点またはかぎ括弧ということにする．［終了］など部の区切り記号か現れるか，ソースプログラムの終わりで，処理部は終了する．これらを“部終了表示子”という．

個々の文節の終わりは，エラーケースも含めて考えると，句読点か，右かぎ括弧か，部終了表示子である．正しいプログラムならば，かぎ括弧や部終了表示子は句読点の直後だけに現れる．句読点なしでこれらが現れたならば，左かぎ括弧のときは直前に読点を，右かぎ括弧または部終了印字子のときは句点を仮定する．右かぎ括弧の次に句点がないときも，句点を仮定する．これにより，「句読点が現れたら，文節が終わる」と考えてよいことになる．

4.10.3 予約語・名前の選択規則

まほろばでは，予約語が文脈により決まるので，それが予約語なのか利用者定義の名前なのかは，文節を解析しないと判断できない．コンパイラは，利用者定義の名前かどうかを，名前表を調べて判断する．以下の例は，判断が難しい例である．これらの判断のために，次の順序の選択規則を設けた．

(1) 句点で終わる文節中に代入記号（＝，→，←）がある場合は，文末の予約語は調べずに，代入文とみなす.
(2) 文節の最後の予約語は，長い方を優先する.
(3) 文節の先頭の名前で二つ以上候補があったときは，長い方を優先する.
(4) 名前の選択は構文解析までで決定する.

例１：からがからからがががからからがまで，次を繰り返す.

呪文のような文節だが,“から”,“からからがががから”，および“が”の

三つが，名前表に登録されていれば，文法的には次のように正しい文節である．

からがからからががからからがまで，次を繰り返す．

まほろばの助詞が文脈により予約語となったり，名前（の一部）になる例である．

例２：Ａ＝Ｂより大きいのときは，

“ ＝ ”と，“ より大きい ”の二つの比較演算子があり，エラーとみなせそうだが，“ Ｂより大きい ”が名前表にあれば，文法的には正しい文節である．

例３：Ａ＝Ｂを足す．

日本語の足し算の構文中に“ ＝ ”があり，エラーとみなせそうだが，“ Ｂを足す ”が名前表にあれば，文法的には正しい文節である．ないときは，“ ＝ ”があるので，Ｂが定義されていてもエラーである．ここでは，代入記号を優先する上記の規則（1）が適用されている．

例４：その他の場合，

“ その他 ”が変数として定義されているときは，“ その他 ”という名前の変数の場合なのか，OTHER の意味なのか区別できない．これを解決するために，(2) の規則を設けた．こうすると，後者がまほろば構文として採用される．

例５：ＡがＢがＣのときは，

“ ＡがＢ，ＢがＣ，Ａ，Ｃ：整数 ”が定義されているとする．このとき，“ Ａ＝Ｂが Ｃ ”なのか，“ ＡがＢ＝Ｃ ”なのか二つの解釈が成り立つ．これを解決するために，(3)の規則を設けた．これを適用すると，左側の長い方の名前が採用されて，“ ＡがＢ＝Ｃ ”と解釈できる．

例６：ＡがＢがＣのときは，

“ ＡがＢ，ＢがＣ，Ａ，：整数 ”と“ Ｃ：文字 ”が定義されているとする．規則（3）により“ ＡがＢ＝Ｃ ”が採用されるが，コンパイラの意味解析で，整数型と文字型の比較はデータ型不一致と判定され，エラーとなってしまう．“ Ａ＝ＢがＣ ”と解釈していれば，データ型不一致にはならない．ただし，規則（4）の「名前の選択は構文解析までで決定する」を適用するので，構文解析ではここまでの整合性のチェックは要求しないものとする．

4.10.4 文節の分類

字句・構文同時逆向き解析方式では，文節分類表を用意し，文節を文末の予約語と句読点を中心に分類している．この抜粋を，**表 4.1** に示す．ここで，オペランド数とは“ AをBに足す．”ときのA，Bの個数のことであり，分離助詞とは，オペランドを区切る“ を ”のことである．レベル１の文字として，代入記号や比較演算子を含むかどうかの項目も設けた．

表 4.1 文節分類表（抜粋）

文節種別	文末予約語と句読点	オペランド数	分離助詞	レベル１記号
代入 A	.	？	－	←
代入 A	.	？	－	＝
代入 B	.	？	－	→
範囲指定の繰返し	ずつ変化させて,	4	を，から，まで	なし
脱出	ぬける.	0	－	なし
呼出し	を呼ぶ.	1	－	なし
代入 1	を入れる.	2	に	なし
代入 2	に入れる.	2	を	なし
条件 A	のときは,	2	－	比較演算子
条件 1	のときは,	2？	が	なし
以外は	それ以外は,	0	－	なし
条件 2	でないときは,	2？	が	なし
繰返し	次を繰り返す.	0	－	なし
条件 3	より大きくない.	2	が	なし

4.10.5 基本アルゴリズム

字句・構文同時逆向き解析方式の基本アルゴリズムを次に示す．

(1) ソースプログラムを入力し，句読点まで文節をスタックする．

① スタックをしながら, その途中にレベル１の代入記号か比較演算子があるか調べる．

② 文末が句点で，レベル１の代入記号があるときは，代入文と決まる．

(2) 代入文と決まらないときは，その文節が文節分類表のどの行に該当するかを調べる．

① 文末の字句が文末分類表の“ 文末予約語と句読点 ”と一致するか，長いものから順に調べる．

② 同じ“ 文末予約語と句読点 ”が登録されている場合がある．“ のときは, ”などの条件の文節であり，これは,“ レベル１記号 ”欄の比較演算子の有無で，どの行に該当するか区別する．

③ 該当する行がなければエラー文節である．

(3) 文節分類表のオペランド数により処理が分かれる．

① 0，1 または？の場合は，LL(1) 文法が適用できる文節なので，文節種別対応に，先頭から通常のコンパイラの構文解析処理と同じ処理をすればよい．

② 2 の場合は，(4) の試行錯誤処理を行う．これが不成功のときはエラーである．

③ 3 個の場合は, 右端から左方向に, 2 番目の分離助詞と一致するレベル１のものを探す．「一致するものがあれば, その分離符の左側の部分について，(4) の試行錯誤処理を行う．」その処理が不成功のときは，さらに左方向に 2 番目の分離助詞を探し，うまくいくまで，「…」を繰り返す．うまくいかずに文節の左端に達すると，エラーである．

④ 2？のときは，条件文節だが，構文が複合条件かどうか，その詳細が未定である．これを１つの条件ごとに分解していく．途中で (4) の試行錯誤処理を必要とするときもある．④の詳細は，省略する．

(4) 試行錯誤処理：“ ＡをＢをＣ ”と“ を ”のように，文節部分と分離助詞が与えられている．右端から左方向に，分離助詞と一致するレベル１のものを探す．「一致すれば, その左側と右側が名前表に登録されているか調べる．」いずれかが登録されていなければ，さらに左方向に分離助詞を探し，一致するまで，「…」を繰り返す．一致せずに，文節の左端に達すると，分解が

不成功としてもどる.

(5) (3)③や(4)で，助詞の前後が定数だったり，小括弧で囲まれた式だったりする場合がある．それらは，名前表を探すのではなく，その部分に対し LL(1) 文法の構文解析を左側から適用する.

4.11 評価

まほろば言語の設計時に定めた設計方針は次の四つであった。本節では，これらが達成できているかどうかを評価する．ただし，この中で(4)については，Java に GOTO 文やポインタ変数がなく，COBOL にもポインタ変数がないことにより，他の言語よりも信頼性の高いプログラムを書くことができると評価され, それが一般にも認められているので, 本論文ではこれ以上は議論しない.

(1) 日本語として違和感がない.
(2) データ構造とアルゴリズムが記述できる.
(3) 事務処理分野のプログラムが記述できる.
(4) プログラムの品質に悪影響を与える機能は，極力サポートを控える．GOTO 文やポインタ変数は少なくとも当初は用意しない.

4.11.1 日本語として違和感がないか

この項目については定量的な評価はできていない．ここでは，若干の人数の記述実験での感想を紹介し､日本語構文が現在のようになった経緯を説明する.

学生 5 人（情報工学を専門とする工業専門学校の 5 年生）と社会人（SE）10 人にまほろば言語について半日のプログラミング教育を実施した. 学生はいずれも C 言語に習熟しており，社会人は COBOL 言語に習熟していた．彼らのまほろばに対する評価は，覚え易く読み易くなっていることを認めつつも，if-then-else という構文の方が視覚的に分かり易いという意見が多く, なぜわざわざ日本語でプログラムを書く必要があるのか，C や COBOL で十分ではないかという意見が大勢であった.

まほろば言語のコンパイラが未完なので，まほろば言語の生産性・信頼性・保守性のデータを取得できていない．それらのデータを蓄積し，既存プログラム言語と比べ数値でそれが優れていることを説明しない限り，既存プログラム言語に慣れ親しんでいる経験者に日本語プログラム言語の良さを理解させ，それを使うようにさせるのは難しいと考える．

次に，日本語表現の多様さに対しどのような方針で仕様設計し，現在の日本語構文になったかについて説明する．

当初のまほろばの言語仕様では“日本語らしさ”を強く意識したため，現状と比べ次のように多くの構文をサポートしていた．

(1) 代入：…を…に入れる．…に…を入れる．（移すという書き方も可）
(2) 演算：…を…に足す．…に…を足す．…を…から引く．…から…を引く．
…を…に掛ける．…に…を掛ける．…を…で割る．…で…を割る．
…と…を…に足す．
(3) 比較条件：…が…より大きいときは，　…が…より小さいときは，
…が…と同じときは，　…が…のときは，（最後の“は”は省略可）
(4) 繰返しの範囲：　…が…から…まで，…ずつ変化させて，次を繰り返す．
(5) 予約語で漢字の書き方も仮名の書き方も許す．・・・

このように多様な表現を許していたので，自分自身ですら，条件文記述時に“ときは”と書くべきところを“ならば”などの類似表現で書いてしまうということをしばしば経験した．また，語順が“…を…に足す”だったか“…に…を足す”だったか迷ったり，仮名の“たす”か漢字の“足す”のいずれがよいか迷ったりして，仮名漢字変換の変換キーの前で指がとまるということも多かった．これらは日本語としていずれも許されており，通常の日本語を使うときには注意しないで使っているからである．

当初は，これら迷った事例のいずれも書けるようにする，すなわち，自然に頭に浮かんだ“日本語”をできるだけ制約しないで使用可能とする方針であった．結果として，多様な表現は片端から追加するという羽目となり，同一機能に対し多くの書き方を許すことになってしまった．たとえば，“…が…と同じ”

があるのだから“…が…と違う”も追加するなど，次々と書き方が増えてしまった．

どれを許しどれを許さないかの基準を設定するのが難しく，この方針のままだと際限がなく書き方が増えてしまう可能性が強いので，方針変更し，使用頻度の高い単純代入, 加算, 減算, 及び大小比較だけをサポートすることにした．この結果，覚えられる範囲の限定した数の日本語構文となり，プログラム作成時に迷うことがほとんどなくなり，際限のないコンパイラの拡張もなくすことができたと考えている．

このように試行錯誤したのは，日本語らしいとは何かの判断基準を設計開始時に設定しなかったことによる．現段階もこの説得力のある判断基準を明示できていないが，これは早期に文書化する必要がある．

4.11.2 データ構造とアルゴリズムの記述能力

4.11.2.1 ミニコンパイラによる記述実験

プログラム言語の実用性を評価するとき，重要な評価項目の一つにそれで通常のデータ構造とアルゴリズムが記述できるかどうかがある．コンパイラでは多くのデータ構造と各種のアルゴリズムが使われている．コンパイラを評価対象のプログラム言語で記述することができれば，一定レベルのデータ構造とアルゴリズム記述が可能であると評価できる．これについては前例があり, Wirth が PASCAL 言語を設計したとき，PASCAL コンパイラや PL/0 言語コンパイラ[Wirth79]を PASCAL 自身で記述して，それがアルゴリズム記述に適していることを実証した．

日本語プログラム言語“まほろば”の言語仕様の実用性評価のために，それでコンパイラを記述することを試みた．まず，まほろばのサブセット言語仕様である“まほろば 0”の文法を定めた．まほろば 0 の言語仕様は Wirth の PL/0 言語および中田の PL/0’言語[中田 96]を参考にし，データ型や定数の種類や文の種類はそれらとほぼ同等な記述能力を持つようにしたが，関数（PL/0 ではプログラム）のもどり値，ネストおよび再帰呼出しはなく，出力は標準関数でのサポートとした． LL(1) 文法適用可能な文法とするため，条件は小括弧で囲むという制約を設けた．まほろば 0 のソースプログラムで記述できる文字は 2

バイトの漢字符号系だけとし，行と空白の規則はまほろばと同じとした．名前や構文は日本語構文であり，名前にも漢字や仮名が使用できる．

このまほろば 0 言語処理系をフルセット仕様のまほろばで記述実験し，まほろばにデータ構造とアルゴリズム記述能力があるかどうかを調べた．まほろば 0 コンパイラは，1 パス・コンパイラであり，“ まほろば 0 仮想計算機 ” の機械語の目的プログラムを生成後，中間言語をインタプリタで実行する．この記述実験で作成したまほろば 0 処理系とそのソースリストは**付録 B** で示す．

4.11.2.2　まほろば言語仕様の中で使用した機能と不足だった機能

まほろば 0 処理系では，まほろばの次の言語仕様を使用している．この範囲で，処理系が必要とするアルゴリズムは記述可能であった．

(1) データ型：整数，文字，レコード型，領域共用型，および列挙型．利用者データ型は多用している．実数，10 進数，数値編集，範囲型は使用していない．
(2) 空文以外はすべて使用している．
(3) 関数の入れ子は使っていない．プログラム先頭の共通の名前定義と，変数の常駐指定は使用している．
(4) 四則演算の使用頻度は，“ A に B を足す ” と “ A から B を引く ” の形式がほとんどで，他の書き方は “ A＋B→C ” を若干使用している．
(5) 入出力標準関数：ソースプログラムをテキストファイルとして入力するため，ファイルオープン，ファイルクローズ，ファイル入力の関数を使用している．ソースファイル名の取得とエラーメッセージ出力のため処理系使用者との交信用に標準入力と標準出力関数を使用している．COBOL で通常使う順ファイルや，相対ファイル，索引ファイルは使っていない．
(6) その他の標準関数：異なるデータ型に変換する文字整変換関数と整数文字変換関数，および先頭の番地と長さを指定して転記する可変長代入関数を使用している．この他の標準関数は使用していない．

この処理系の記述開始時のまほろば言語の言語仕様には，領域共用型，プロ

グラム先頭部分の関数間共通の名前定義, 変数の常駐指定（スタティック属性）はなかった．これらは，次の理由により追加した．なお，一般のコンパイラではポインタ機能が多用されているが，これを使いたい箇所は配列で済ますことができた．

(1) 領域共用型： 領域共用や再定義の機能は，データの安全性からは問題があるので当初はサポートしていなかった．まほろば 0 コンパイラの目的プログラムは，命令識別部（命令コード）とオペランド（番地，値）の両方が入る整数の配列として定義したが，前者の命令識別部は列挙型として扱った方が保守性と文書性が優れる．このため，“ 異なるデータ型が同一領域を共用する機能 ” を追加した．
(2) 関数間共通の名前定義：関数間の共通の名前は，関数を入れ子にし上位階層の関数中に広域の名前として定義した．結果として，上位階層の関数中に，自分自身が使う変数などの名前と，自分の関数に関係のない下位階層間の関数どうしが使う名前とが混在し，関数としての独立性が欠けてしまった．これを是正するため，プログラム先頭の関数の前に共通の名前を書けるようにしたことで，個々の関数の独立性を増すことができた．
(3) 変数の常駐機能：一般には，関数の変数は関数からぬけると再利用できない．ソースプログラムの入力行などは一つの関数だけで参照する変数であっても，再利用のため残しておく変数である．残しておくために，プログラム先頭の共通変数として，扱わざるを得なかった．これは，一つの関数しか使わないのに，共通の名前となっており，意図せずに他で使われてしまう危険性があった．変数の常駐指定を可能としたことにより，このような変数のデータ隠蔽ができるようになった．

ミニコンパイラとはいえ，一つのコンパイラをまほろばで記述できたので，“ まほろばはデータ構造とアルゴリズム記述が可能な言語である ” ということができる．

4.11.2.3 今後不足が予想される機能

最終的には，まほろばのフルセット仕様のコンパイラをまほろばで書くこと

を計画している，このためには，次の機能不足が分かっており，これらはできるだけ早く実現する必要がある．

⑴ レコード型の配列の初期値：現在の仕様の範囲で，予約語表などの表に初期値を記述するには，「それぞれの行の項目毎に変数を定義し，そこに初期値を書き，それらをまとめて一つのレコードにする．そのレコードと，レコード型の配列とを，領域共用型として定義する．」という厄介な記述をする必要がある．これを避けるために，文書性の良い“ レコード型の配列の初期値 ”を，それも変数としてではなく定数表として，記述可能とする必要がある．
⑵ 範囲型の条件：“（Ａ≧５）および（Ａ≦１０）のときは ”のように範囲内を判定する書き方が増えるが，“ Ａ＝５～１０のときは ”あるいは“ Ａが５から１０の範囲のときは ”と書けると，文書性がよくなる．

4.11.3 事務処理分野のプログラムが記述可能か

4.11.3.1 COBOL 特有の補助機能がないが，問題にはならない

まほろばでは事務処理で使用可能とするために，10 進数と数字編集のデータ型と，標準関数で順，行順（テキストファイル），相対，索引編成のレコード型のファイル入出力機能を，COBOL を参考にサポートしている．これにより大きな問題は解決したはずであるが，他に問題はないか次の調査を行った．

2 冊の標準 COBOL の入門書[西村 93][今城 90b]中に掲載されているプログラム例題について，まほろばで記述可能か調査した．例題は 17 本あり，12 本は記述できたが，5 本は記述できなかった．記述できなかったのは，次の理由による．いずれも，COBOL で昔からある補助機能が使われている．これらは独立した大きな機能であり，使うと便利であるが，コンパイラ作成者の負担も大きい．事務処理向けのプログラム言語として，それらの機能がなくても，問題にはならないと考える．

⑴ ソート機能（2 本）：この機能は一般には，ソートユーティリティで代替できるので，プログラム言語として必須機能ではない．ただし，多くのプロ

グラマが COBOL ソートを使っており，根強い人気がある．

⑵ 報告書作成（リポートライタ）機能（1 本）：通常のレコード定義と出力文で代替記述可能だが，量が大幅に増え，アルゴリズムもそれなりに大変なので，まほろばではプログラミングしなかった．国内では，ほとんど COBOL の報告書作成機能は使われていないので，この機能がなくても弱点にはならない．

⑶ 通信（オンライン）機能（1 本）：標準 COBOL の通信機能は，1970 年代初頭の旧式のオンライン制御プログラムを前提としており，次の規格改訂では削除されることになっている．現実の COBOL オンラインのプログラム文法は，ベンダ毎に異なり互換性がない．標準 COBOL の通信機能をまほろばが持っていなくても，弱点にはならない．ただし，この分野の各ベンダ独自のオンライン制御プログラムや画面定義／GUI 定義プログラムは，COBOL や C などの言語を意識したサポートをしている．それらにまほろばを認知してもらうために，まずまほろばの市場性を高めることが必要である．

⑷ 標準組込み関数（１本）：標準偏差や，平均値，最大値など COBOL の標準関数を使っている．組込み関数は 1992 年に国際標準になったが，ほとんどの COBOL プログラマはこの機能を使ったことがない．まほろばの標準関数は，標準プログラム言語でよく使われる組込み関数は順次サポートしていく方針である．

以上により，まほろばで事務処理記述が可能であることが実証できたが，次の機能不足も判明した．

4.11.3.2 不足の機能

COBOL 学習書の例題でまほろばで記述可能だった 12 本の中に，まほろばにあれば便利なものがいくつかあった（下記の(1)～(3)）．これらは現在のまほろばの機能で代替表現できるが，いずれサポートしたい機能である．今回の 17 本の中では使われていなかったが，次の (4)～(5) は，COBOL プログラマが多用しているので，まほろばで事務処理を本格的に記述する場合には必須である．

(1) INITIALIZE 文：レコード型に含む下位の変数をデータ型対応にゼロや空白などで初期化する.
(2) 名前なしの変数：ファイルのレコードに含まれるが，当該のプログラムには無関係の変数に対し，名前はつけずに大きさだけ指定する機能.
(3) 長さが相手によって変わる標準定数名：SPACE, ZERO, HIGH-VALUE, LOW-VALUE など.
(4) COPY 文：C 言語の#include 相当．共用するソースをコンパイル時に取り込む機能.
(5) STOP 文：副プログラムからのプログラム終了.

4.12 まとめ

本章では，分かち書きのない本格的な日本語プログラム言語 “ まほろば ” の文法を定義し，それらが次の設計目標が実現できているかを考察した.

(1) 日本語として違和感のない仕様となっているか.
(2) データ構造とアルゴリズム記述の適性があること.
(3) 事務処理プログラムが記述できるか.

今後はまほろば処理系を完成させるとともに，次の課題にも取り組みたい.

(1) 2003 年からは，高校の普通科で情報科目が必修となる．ここでのアルゴリズム記述言語としての適用可能性の検討.
(2) オブジェクト指向機能の追加．C++，Java，COBOL 次期規格の当該仕様を意識した仕様を策定要.
(3) C/C++，Java，COBOL の三つの既存環境への変換仕様の策定．これらの言語への変換機能をまほろばコンパイラのオプションとして支援できれば，既存の実行環境を享受できる.
(4) C/C++, Java，COBOL からまほろば言語への変換仕様の策定とそれらのコンバータの作成. この変換結果はプログラム仕様書としても活用できる. このとき, 利用者の定義したアルファベットの名前を日本語の名前に変換するとより効果的である.

第 5 章　結論

本研究は，筆者らが約 15 年にわたり研究してきたプログラム言語の日本語化と国際化についてまとめたものである．この研究を推進した筆者の基本的な考え方は，ソフトウェア基本構造は英語をベースにしており，英語を母語としていない人々にとって出発点から差があり，公平でないという点にあった．これはプログラム言語においても同様である．COBOL は古いといわれながらも広範囲に根強く英語圏で使われているのは，COBOL は彼らが小さいときから親しんできた英語、すなわち母語をベースにしており，母語で思考し表現した業務仕様を素直にプログラムに反映できることが，大きな要因である．これと同じことを非英語圏でも享受できるようにしたいというのが筆者の問題意識であった．この問題意識は業務処理の分野では，“ 国際化・現地語化モデル ”という概念で標準化の世界の共通認識となったが，プログラム言語自身では確立した共通認識になっていない..

この問題意識に基づき，プログラム言語の日本語化と国際化について次の三つの分野で筆者が行った研究を本論文でまとめた.

(1) COBOL 国際化機能の標準化の推進

COBOL 日本語機能の言語仕様を開発し，これをマルチオクテット文字機能さらには国際化機能として一般化した．その成果を国際規格として提案し，採用されることが決定している．標準化にあたっての課題は，日本という地域性を排除して，国際的に通用する技術とすること，および日本語機能を用いた既存資産の移行性を確保することの二つだった．次の方針を策定実行して，将来にわたって使用可能な汎用的な国際化機能の言語仕様を制定することに成功した.

(a) 対象とする文字をアルファベット 26 文字以外の世界各国の文字に拡張するとともに，多くの符号系に対応可能な普遍的な枠組を設定する.

(b) 既存の文字型に加えて，漢字などの文字型を新設する．それらの型を処理する文は同等とし，同じ業務プログラムを各国ごとに適用するときには，データ定義だけを修正しロジックの変更は不要とする仕様とする.

(c) 国内で普及している日本語処理機能を，国際化機能の核として採用する．

(2) 予約語まで日本語化した EAGLE/4GL の開発と普及（分かち書きレベル）

10 年ほど前に，第 4 世代言語（4GL）はメインフレームの業務アプリケーションを効率よく構築する手段として注目を浴びていた．我々が開発した EAGLE/4GL は，その構文に日本語（分かち書き）を採用したことにより，プログラム作成が楽になるとともに保守性を大幅に向上させることができ，万を超えるユーザに利用された．また，メインフレームのオンライン制御プログラム環境（OLTP 環境）と対話環境（TSS 環境）の両方で動作可能なアプリケーションを作るため, 環境独立なオンライン業務プログラム処理方式を採用した．現在，EAGLE/4GL はアプリケーション開発の主流ではないが，そのテクノロジは現代のオープン環境にも有効に適用できる．

(3) 分かち書きをしない本格的な日本語プログラム言語 “ まほろば ” の開発

分かち書きをしないより日本語に近い日本語プログラム言語 “ まほろば ” の言語仕様を設計した．この言語の設計方針は次の四つとし，記述実験を行い，いずれの項目もほぼ達成していることを実証するとともに，今後の課題を指摘できた．

(a) 日本語として違和感のない仕様とする．
(b) 標準プログラム言語で記述できる “ データ構造とアルゴリズム ” を記述可能とする．
(c) 言語普及を目指すために，市場が大きい事務処理分野で適用可能とする．
(d) プログラム品質に悪影響のある GOTO 文とポインタ変数は支援しない．

これらの研究は，プログラム言語の国際化の主要な三つの発展段階に対応し，それぞれの段階で代表的なものの一つとして，各分野に大きな影響を与えてきた．さらに今後のこの分野の研究に寄与することが期待される．

それぞれの今後の課題は次の事項である．

(1)　COBOL などの標準プログラム言語

(a) 英数字と漢字などの国別文字は異なるデータ型として扱われ，同一データ項目の中には共存できないが，入力データなどでは混在することも多い．整合性のとれたこれらの混在データ型が必要である．

(b) プログラム中では同時には国別文字は一つしか使用できない．同時に複数の国別文字を使用したいという多国語環境に対応した仕様が将来必要となる．

(c) COBOLの国際化機能は1990年代のスタンドアロンのシステムモデルを前提にしている．これは，複数の計算機が同時にインターネット環境で司令塔なしで協力しながらそれぞれ自律して動作する今後のシステムモデルには限界がある．このような複雑なシステムに適応する国際化のシステムモデルの再構築が必要である．

(2)　EAGLE/4GL

製品としてはすでに旬は過ぎているので，そこで開発されたテクノロジ“環境独立オンライン業務プログラム処理方式”を現在主流のオープン環境，特にEJB 環境で他の言語でも適用可能なように一般化することと，その支援ツールの開発が課題である．

(3)　日本語プログラム言語まほろば

まずは次の(a)〜(c)を実現し，実用化することが課題である．このためには開発費の確保が必要である．次に(d)〜(g)の機能拡張が課題である．

(a) まほろば処理系の完成

(b) 実証実験で指摘された不足機能のサポート

(c) 生産性・信頼性・保守性データの蓄積により，この言語が優れているという説得力の確立

(d) 高校の情報科目でのアルゴリズム記述言語としての適用可能性の検討

(e) オブジェクト指向機能のサポート

(f) C++，Java，および COBOL の三つの環境への変換仕様の策定と支援系の作成.

(g) C/C++，Java，COBOL からまほろば言語への変換仕様の策定とそれ

らのコンバータの作成.

COBOL と EAGLE/4GL は当初の目標を達成することができ, まほろばについては言語仕様の実用性は実証できた. 今後の課題は本格的な処理系を開発し, 日本語を母語とする人々に “ 違和感のないプログラミング ” が可能となるようにしていくことである.

付録 A　文献解題：プログラム言語の日本語化と国際化

A.1　はじめに

1979 年から 1980 年にかけて，ほとんどのメーカが日本語処理システムを実現したことにより, 日本語データ処理が本格化した. 各企業の情報処理部門は, エンドユーザへのサービス向上のため日本語表示・印刷のシステムを次々と構築したが, 彼らの自然な要求は, 仕様記述やプログラミングにも日本語を用い, 自分たちもコンピュータの日本語処理の恩恵に浴したいということだった．この要求に対応し, 1980 年代後半に日本語プログラム言語の研究と商品化が盛り上がりをみせた．情報処理学会の全国大会や各種の研究会でも，この時期に日本語プログラミング関連の発表がピークとなっている．

1990 年代になると，この勢いが急激に衰え報告の数は激減したが，1998 年に出版された次の 4 冊の単行本は日本語プログラム言語に言及している．

[玉井 98]　玉井哲雄（東大）：コンピュータの言語，新・知の技法（小林康夫, 船曳建夫（東大）編），pp.151-163，東京大学出版会，1998.
[黒川利明 98]　黒川利明（日本 IBM）：プログラム言語の仕組み，170 ページ，朝倉書店，1998.
[中所 98]　中所武司（明大）：ソフトウェア工学, 196 ページ, 朝倉書店, 1998.
[本位田 98]　本位田真一，大須賀昭彦（東芝）：オブジェクト指向からエージェント指向へ，ソフトバンク，1998.

同じ年の 1998 年に，情報処理学会の秋のプログラミング・シンポジウムが「日本のプログラミング」をテーマに開催され，日本語プログラミングの発表が六つあった．さらに近頃はこの分野で注目すべき研究が学会の研究会や論文誌などでに報告されている．これらのことは，英語をベースにした現在のプログラム言語に対し，底流に違和感を覚えており，日本語プログラム言語への期待が根強いことを物語っているといえよう．

[プロシン 99]　秋のプログラム・シンポジウム「日本のプログラミング」報告

集 1998 年 9 月 16 日～18 日，情報処理学会，1999.

本付録では，日本語プログラム言語に関係する今までの研究と商用化の約 175 件の文献を紹介し，日本語プログラム言語がどのように発展し現状がどうなっているかについて述べ，今後のこの分野の研究と開発の参考とする．54 件の重要な文献には下線を引いた．

文献調査は，1979 年以降の情報処理学会の論文誌，全国大会及び研究会報告（ソフトウェア工学研究会とプログラミング研究会（その前身を含む)），電子情報通信学会論文誌，コンピュータソフトウェア誌を主な対象とした．全国大会の発表については，論文誌や研究会でそれと関連する報告がないものを中心に記載した．一般のコンピュータ雑誌，企業の技術論文誌及びマニュアルについては，ほとんど参照できなかった．

本付録の構成は次のとおりである．**A.2** では，日本語プログラミング全般の文献を載せる．つづいて，日本語プログラム言語を次の三つの分類に従い，**A.3**，**A.4**，**A.5** で説明する．**A.6** でまとめを述べる．

(1) Fortran，COBOL など標準プログラム言語での日本語データ処理と識別名の日本語化．
(2) さらに進め，予約語まで日本語化した“ 本格的な日本語プログラム言語 ”．
(3) 日本語による仕様記述言語．

A.2　日本語プログラミング全般

A.2.1　総括的な文献

いくつかのコンピュータ関連辞書で，日本語プログラム言語に言及しているが，全体の中ではわずかの分量であり，日本語プログラムが期待に反し，普及していないことを物語っている．

[岩波辞典 90]　岩波情報科学辞典，p.551 及び p.720，岩波書店，1990.
[和田英穂 91]　和田英穂（富士通）：プログラム言語の将来，コンピュータの

事典第 2 版，pp.430-432，朝倉書店，1991.
[共立辞典 94]　日本ユニシス編:共立総合コンピュータ辞典第 4 版, pp.740-741, 共立出版，1994.
[アルゴリズム辞典 94]　アルゴリズム辞典，pp.587-589，共立出版，1994.
[ハンドブック 91]　システムエンジニアリングハンドブック，pp.407-409，オーム社，1991.

日本語プログラム言語の研究・開発が本格的に始まった頃に，その意義と期待を述べたのが[神田 78]であり，次のように主張している．

“ プログラムを含む計算機の利用技術としてのソフトウェアは高次の概念，複雑な処理アルゴリズムについての思考を必要とする．日本人のソフトウェア設計者／作成者が思考し，思考の結果を他人に見せられるように記述するには日本語を用いて行うのが最も自然である ”

1986 年から 1987 年にかけて岩波書店の雑誌“ 科学 ”に連載された論稿をまとめた[黒川利明 90]の第 2 章“ 日本語で考えええる ”には，日本語でプログラムすることの意義が述べられている．長くなるがそれから引用する．なお，この中で言及している朱唇については **A.4.1.2** で説明する．

“ それでは，日本語をベースにしたプログラム言語の利点は何か，といえば，それは「誰にでも読める」ということだろう．あるいは，プログラムのもつ言語のセンスというものを率直に表現できるということだろう．クヌースが，『文芸的プログラミング』の中で主張していたことだが，本物のプログラマは変数の名前の一つ一つに，随分と気を使うものである．そういう変数名の文脈が英語風だとしたら，日本人のプログラマは，変数名を付けるのに日本語のセンスをどう生かせるものだろうか．

ことばを学ぶことが，多くの学問での入門課程であるというのは，ことばによってのみ諸概念を表現し，他人に伝達することができるからである．それだから，個々の単語を含めたことば全体を大事にしなければいけないというのはわかりきったことだろう．その姿勢をプログラミングの場でも貫くとすれば，それは日本語に基づいたプログラミング言語の開発になるのではないかというのが，ここでの主張の要約である．裏返せば，日本語に基づいたまともなプログラミング言語の提案が少なすぎるので面白くない，ということになるので

あるが，ともかく《朱唇》の紹介を試みることにしよう．”

予約語まで日本語化したときの構文については，[渡辺勝正 80]が考察している（**図 A.1**）．[石田 83]は，ソフトウェア文書としての観点からプログラム言語の予約語と識別名の日本語化と仕様書の自動生成について論じている．[草薙 85]では，言語学の観点から制限日本語を考察している．

[神田 78] 神田泰典，杉本正勝（富士通）：ソフトウェア工学における日本語の役割，情報処理学会ソフトウェア工学研究会資料(5-1)，pp.1-9，1978-1-26.
[渡辺勝正 80] 渡辺勝正，都司達夫（福井大）：日本語プログラム言語の開発にむけて，情報処理学会第 21 回（昭和 55 年）全国大会，5I-1-1，pp.1011-1012，1980.

```
プログラム　素因数分解　のはじまり
  ここで使用する名前は
      整数　数，底，べき
  ここからくりかえす
      数←ソース入力
      もし　数＝０　ならば　リスト←“ これでおわり ”，改行
      さもなくば
          ここから
              リスト←数：４，“ ＝１ ”
              底←２
              条件　底≦数　の間
                  べき←０
                  条件　余り（数，底）＝０　の間
                      数←数／底
                      べき←べき＋１
                  ここまでくりかえす
                  もし　べき＞０　ならば
                      リスト←“ ＊ ”，数：３，“ ＊＊ ”，べき：２
                  底←底＋１
              ここまでくりかえす
          ここまで
      リスト←改行
  ここまでくりかえす　条件　数＝０　まで
プログラム　素因数分解　のおわり
```

図 A.1　予約語まで日本語化したときの構文　[渡辺勝正 80]より

[石田 83] 石田晴久（東大）：ドキュメント作成の現状と将来，電子通信学会誌（ソフトウェア生産技術の現状と将来特集号），Vol.66，No.4，pp.402-407，1983.
[草薙 85] 草薙裕（筑波大）：自然言語とコンピュータ言語，216 ページ，講談社サイエンティフィク，1985.
[黒川利明 90] 黒川利明（日本 IBM）：作品としてのプログラム，191 ページ，岩波書店，1990.

[機シ振協 88]は，1987 年までの日本語プログラミングの総括的な調査及び記述実験報告であり，文献リストには 103 件の記載がある．この調査は，玉井（当時，三菱総研）が中心に行った．[玉井 90]]は[機振協 88]の調査報告書の要約とであり，[玉井 88]はそこでの記述実験結果と日本語プログラム言語の展望を述べている．[今城 99ａ]と[今城 99ｂ]は筆者によるもので，前者は本付録の元になった文献リストであり，後者では日本語プログラミングの発展段階と言語設計の留意点を述べている．

[機シ振協 88] 機械システム振興協会（委託先：三菱総研）：21 世紀における新社会システムに関する調査報告書―日本語プログラミングの調査―（62-R-6(2)），168 ページ，1988.
[玉井 88] 玉井哲雄，小久保岩生，牧野京子（三菱総研）：日本語プログラミングに関する実験と考察，日本ソフトウェア科学会第 5 回大会論文集，B2-1，pp.81-84，1988.
[玉井 90] Tetsuo Tamai（筑波大）：On Japanese-based Programming, Journal of Information Processing, Vol.13, No.1, pp.49-56,1990.
[今城 99a] 今城哲二（日立／奈良先端大）：日本語プログラム言語文献ノート，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.9-16, 1999.
[今城 99b] 今城哲二（日立／奈良先端大）：日本語プログラム言語の設計，第 40 回プログラム・シンポジウム報告集，情報処理学会，pp.7-19，1999-1-12.

A.2.2 日本語プログラミングの環境と効果

日本語プログラミングには，それを実践するための環境と言語処理系が前提

となる．東京農工大ではその研究を精力的に行い，学内で利用している．

[高橋 89]　高橋延匡（東京農工大）：日本語プログラミング環境，情報処理，Vol.30，No.4，pp.363-372，1989.
[高橋 88]　高橋延匡：研究プロジェクト総説：OS/omicron の開発，情報処理学会オペレーティングシステム研究会（39-5），14 ページ，1988-6-17.
[鈴木茂夫 89]　鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡（東京農工大）：OS／omicron における日本語プログラミング環境，情報処理学会論文誌，Vol.30，No.1，pp.2-11，1989.
[並木 92]　並木美太郎，早川栄一，下村秀樹，田中泰夫，中川正樹，高橋延匡（東京農工大）：言語 C コンパイラのマルチバイト化の実現方式，情報処理学会論文誌，Vol.33，No.11，pp.1331-1340，1992.

日本語プログラム言語の普及のためには，客観的で定量的な効果が最も説得力があるが，そのような論文の数は少ない．次の中川の一連の論文は，C 言語の識別名を日本語記述したときの評価報告であり，貴重である．

[中川 92a]　中川正樹，玉木祐二，早川栄一，曽谷俊男（東京農工大）：母国語プログラミングへの方式，実践とその効果，情報処理学会ソフトウェア工学研究会資料(85-2)，pp.7-14，1992-5-21.
[中川 92b]　中川正樹，早川栄一，玉木祐二（東京農工大）：日本語プログラムの可読性の評価と検討，情報処理学会ヒューマンインターフェース研究会資料(43-1)，pp.1-8，1992-7-9.
[中川 94a]　中川正樹，早川栄一（東京農工大），玉木祐二（東芝），曽谷俊男（日本 IBM）：日本語プログラミングの実践とその効果, 情報処理学会論文誌，Vol.35，No.10，pp.2170-2179，1994.
[中川 94b]　中川正樹，早川栄一，並木美太郎，高橋延匡（東京農工大）：母語プログラミングの理念, 実現, 実践とその効果, 電子情報通信学会論文誌 D-I，Vol.J77-D-I，No.5，pp.364-374，1994.

日本語プログラミングの実践で最も重要なことは，データ項目の名前（識別

名）を適切に命名し，保守工程の従事者のために，読みやすく，誤解のないコーディングをすることである．次の二つの論文は日本語プログラミングを対象にしていないが，データ項目名を日本語にするときに役立つ文献である．

[黒川清 93] 黒川清，中川優，関根純（NTT）：複合語解析技術を用いたデータ項目名称の標準化手法，情報処理学会論文誌，Vol.34，No.3，pp.447-456，1993.
[関根 93] 関根純，川下満，町原宏毅，中川優（NTT）：体系的な DB 構築のための用語辞書を用いたデータ標準化手法, 情報処理学会論文誌, Vol.34, No.3, pp.457-467，1993.

A.3 標準プログラム言語：日本語化から国際化へ

本節では，Fortran，COBOL ，C などの既存言語での日本語データ処理と識別名の日本語化についての文献を中心に紹介する．国際規格にするために，日本語化を一般的にし，国際化というようになった．

A.3.1 カタカナの識別名

1960 年代の半ばから後半にかけて，国産メーカの多くが COBOL コンパイラで，データ名などの識別名にカタカナを書けるようにした．これが，プログラム言語の日本語化で実用化された最初であり, 今でも現場で使用されている．[近藤 64]は，土居が大学 4 年のときの卒業研究で，Fortran の命令語と識別名のカタカナ化について報告している．これは，日本語プログラム言語の文献では筆者が調べた範囲で初出である（**図 A.2**）．ただし，Fortran のカタカナ化は実用化されなかった．[西村 67]は，カタカナ化だけでなく，識別名の漢字化や予約語の日本語化について述べており，今でも参考になることが多い．

[近藤 64] 近藤頌子，原田賢一，土居範久（慶大）：カナ文字 Fortran，情報処理，Vol.5，No.4，pp.222-223，1964.
[西村 67] 西村恕彦（通産省電気試験所）：日本語と COBOL，情報処理，Vol.

８，No.3，pp.157-160，1967.

[COBOL 研 67] 情報処理学会 COBOL 研究会：国産の COBOL コンパイラ，情報処理，Vol.８，No.3，pp.140-144，1967.

```
ＫＣＣ－１　ＦＯＲＴＲＡＮ
        スウレツ　Ａ（10）θ
        ヤレ　10　Ｉ＝1，10θ
10　ヨメ　20，Ａ（Ｉ）θ
20　ケイシキ　（Ｅ15.1）θ
    Ｘ＝－0.98θ
30　Ｘ＝Ｘ＋0.02θ
    Ｐ＝Ａ（Ｉ）θ
    ヤレ　40　Ｉ＝2，10θ
40　Ｐ＝Ｐ＊Ｘ＋Ａ（Ｉ）θ
    アケロ　θ
    カケ　50，Ｘ，Ｐθ
50　ケイシキ　（Ｆ5.2）5（Ｅ15.1）θ
    フゴウ（Ｘ－0.5）30，30，60θ
60　オワレ　θ
```

図 A.2　土居のカナ文字 FORTRAN　[近藤 64]より

A.3.2　国内での日本語処理機能の標準化推進

1979 年以降の本格的な日本語処理システムの中で, 各メーカはプログラム言語に日本語処理を取り入れたが, 仕様が異なり互換性がなかった. これに対し，日本電子工業振興協会の言語標準化専門委員会（委員長：木下（東芝））が精力的に各ベンダの FORTRAN と COBOL の日本語処理機能を調査し，統一仕様案を作成した．FORTRAN については JIS 原案まで作成されたが, 国際規格と異なる JIS を作成すると，海外から非関税輸入障壁とみなされ，当時激化していた貿易摩擦がさらに悪化するとの心配から，制定は棚上げになった．この反省から，日本語機能を“国際化機能”と一般化し，まず各言語の国際規格に追加し，それから国際規格に忠実に JIS にするという日本語処理の標準化方針が確立された．これを契機に，各プログラム言語の国際標準化活動に日本が積極的に参加するようになった．この頃の論文・報告を次に示す.

[木下 85] 木下恂（東芝）：標準プログラム言語における日本語処理，情報処理，Vol.26，No.3，pp.226-232，1985.
[電子協 84] 言語標準化専門委員会：プログラミング用言語の標準化に関する調査－日本語 FORTRAN－，59-C-483，pp.1-109，日本電子工業振興協会，1984.
[電子協 85a] 言語標準化専門委員会：プログラミング用言語の標準化に関する調査－日本語 COBOL－，60-C-506，202 ページ，日本電子工業振興協会，1985.
[電子協 86] 言語標準化専門委員会：プログラミング用言語の標準化に関する調査－日本語 COBOL 他－，61-C-533，日本電子工業振興協会，1986.
[電子協 85b] 言語標準化専門委員会：JEIDA 規格 電子計算機プログラミング言語 日本語 FORTRAN，JEIDA-42-1985，60-C-526，126 ページ，日本電子工業振興協会，1985.
[日本語 FORT86] 日本語 FORTRAN JIS 原案作成委員会：電子計算機プログラミング言語 日本語 FORTRAN JIS 原案，1986.
[西村 86] 西村恕彦（東京農工大）：FORTRAN 日本語処理の JIS 原案，情報処理学会記号処理研究会（36-6）/プログラミング言語研究会（5-6）共催資料，pp.1-4，1986-3-10.
[床分 86] 床分眞一，今城哲二（日立），花田良平（日立ソフトウェアエンジニアリング）：COBOL 言語における日本語機能，情報処理学会第 32 回（昭和 61 年前期）全国大会，5F-2，pp.427-428，1986.
[阿部 87] 阿部昇，市丸数馬，渡辺照洋（富士通）：C 言語での日本語サポート方式，情報処理学会第 35 回（昭和 62 年後期）全国大会，7R-5，pp.901-902，1987.

A.3.3 マルチオクテット文字処理の国際標準化の推進

プログラム言語をはじめとし，情報処理各分野の日本語処理の標準化を横断的に検討し，国際規格化の方向づけをするために，情報処理学会情報規格調査会は日本語機能専門委員会（委員長：池田（京大） ）を 1987 年から 1990 年にかけて設置した．この中で，中田（筑波大）が中心になり「標準プログラム

言語のマルチオクテット文字処理ガイドライン」を作成し，日本からの国際規格提案の先導的役割をはたした．次の資料・報告が国際標準化を方向づけたものである．

[中田 87] Nakata（筑波大）：Requirements for Character Handling in Programming Languages, ISO SC22 会議資料（1987）．（[日本語委 88] に転載されている．SIGPLAN NOTICES Vol.23（1988)の No.1(pp.5)と No.6 (pp.17-18)の Letter 欄にも関連記事が載っている．）
[日本語委 88] 日本語機能専門委員会：情報技術における日本語機能の標準化に関する調査研究報告書，77 ページ，情報処理学会情報規格調査会／日本規格協会情報技術標準化推進センター，1988-3. （1989 年と 1990 年にもこの委員会の報告書が発行されている．）
[標準化フォーラム 88] 情報技術標準化フォーラム「日本語処理の統一的取り扱い」講演資料，76 ページ，情報処理学会情報規格調査会，1988-12-7.
[標準化フォーラム 90] 情報技術標準化フォーラム「プログラム言語とデータベース」講演資料，111 ページ，情報処理学会情報規格調査会，1990-1-23.

A.3.4 各言語での国際規格の制定

日本語処理機能を一般化した国際化機能を各言語の国際規格に取り入れるために，日本は主導的に作業した．C(1990)，FORTRAN(1991)，SQL(1992)，POSIX(1993)，Ada (1995)，Prolog(1995)，ISLISP(1997)，C++(1998)と次々に国際化機能の規格化が実現した．最も標準仕様の制定が期待されていた COBOL は，規格の前身となる CODASYL COBOL で 1992 年に仕様を確定した．この仕様を吸収する次期 COBOL 国際規格は，オブジェクト指向機能など大規模な追加のため時間がかかっており, 2002 年に制定される見込みである．各言語の国際規格に対応し，JIS も順次制定されている．

仕様のレベルは，COBOL，SQL，Prolog，ISLISP，C++がマルチオクテット文字の識別名を実現しているが，他の言語はデータ処理のレベルである．

MUMPS の国際規格には，国際化機能は採用されていないが，JIS には日本語処理機能を追加している．

各言語の国際化機能についての規格や報告を次に示す.

[JISFORT94] JIS X 3001:1994 プログラム言語 Fortran. （対応する国際規格番号は ISO/IEC 1539:1991 である.）
[和田英穂 95] 和田英穂（富士通）：プログラム言語最新情報－Ⅱ 4. 新しい Fortran－Fortran 90－，情報処理，Vol.36，No.4，pp.297-303，1995.
[長谷部 87] 長谷部紀元（図書館大）：プログラミング言語における日本語サポート方式―多言語サポートの国際標準化の脈絡から―，情報処理学会プログラミング研究会資料（12-4），10 ページ，1987-7-24.
[JISC93] JIS X 3010-1993 プログラム言語 C. （対応する国際規格番号は ISO/IEC 9899:1990 である.）
[JISC96] JIS X 3010-1996 プログラム言語 C (追補１). （対応する国際規格は Amendment 1:1995 to ISO/IEC 9899: 1990 C Integrity である.）
[JISSQL95] JIS X3005:1995 データベース言語 SQL. （対応する国際規格番号は ISO/IEC 9075:1992 である.）
[JISPOSIX94] JIS X 3030-1994 移植可能なオペレーティングシステムのインターフェース(POSIX) 第 1 部 応用プログラム向けのインターフェース(API) [プログラム言語 C]. （対応する国際規格番号は ISO/IEC 9945-1:1990 である.）
[JISPOSIX96] JIS 原案 移植可能なオペレーティングシステムのインターフェース(POSIX) 第 2 部 シェルとユーティリティ，1996. （対応する国際規格番号は ISO/IEC9945-2:1993 である.）
[ISOAda95] ISO/IEC 8652: 1995 Information technology – Programming language – Ada Edition: 2 (monolingal).
[筧 95] 筧捷彦（早大），石畑清（明大），西田晴彦（NTT）：プログラム言語最新情報－Ⅱ 5. Ada-9X－大規模ソフトウェア向きの手続き型言語－，情報処理，Vol.36，No.4，pp.304-314，1995.
[ISOProlog95] ISO/IEC 13211-1: 1995 Information technology – Programming language – Prolog Part 1：General core.
[JISISLISP98] JIS X 3012:1998 プログラム言語 ISLISP.（対応する国際規格番号は ISO/IEC 13816:1997 である.）

[ISOC++98] ISO/IEC 14482:1998 Information technology – Programming language – C++. （対応する JIS は 2002 年に発行予定である.）
[今城99c] 今城哲二（日立）：＜標準活動トピックス＞標準化プログラム言語の国際化，情報技術標準 No.44，pp.2-6，情報処理学会情報規格調査会，1999.
[CODASYL94] CODASYL COBOL Committee：Journal of Development 1993, ISDN 0-9621665-1-4, 1994.
[ISOCOBOL2001] ISO/IEC FCD1989：2001 Information technology － Programming language － COBOL, 869 ページ, 2001-01-15. （2002 年春に DIS（国際推薦規格），2002 年末に IS（国際規格）が発行され，IS に対応する JIS は 2003 年度に発行される見込みである.）
[JISMUMPS95] JIS X 3011:1995 プログラム言語 MUMPS. （対応する国際規格番号は ISO/IEC 11756:1992 である.）

Java はサンマイクロシステムズが PAS（Public Available Specification）という手順で ISO/IEC JTC1 に国際標準案を提出することが認められていたが，自らの意思で提案を行わず，私的な JCP という標準化組織で仕様を決めている．[風間 2000] は Java の国際化と日本語処理について詳説している．

[風間 2000] 風間洋一（NTT）：Java プログラミング・ノート 国際化と日本語処理，206 ページ，アスキー，2000.

A.3.5 ISO での国際化ガイドラインの制定

国際の場で，国際化を横断的に検討する組織として，1990 年に ISO/IEC JTC1 SC22/WG20 が新設された．ここで，国際化の枠組みとプログラム言語規格作成ガイドラインの二つの技術報告（Technical Report）を作成した[TR 国際化 2000], [TR 言語規格 2000]．これらは規格化作業だけでなく，新しくプログラム言語を設計するときにも役立つ文書である．エディタは，前者は佐藤（日本 HP），後者は木戸（日本 IBM）と野田（日電）である．[清兼 98]はコード系，C，UNIX を中心とした国際化と地域化に関し詳細に解説しており，巻末には 212 件の文献及びリソースが紹介されている．国際化の研究や作業をするときの必読書である．

[TR国際化2000]　TR X 0030:2000 国際化の基本構造，59ページ，日本規格協会，2000．（対応する国際規格技術報告番号はISO/IEC TR 11017:1998である．）

[TR言語規格2000]　TR X 0031:2000 プログラム言語規格策定の方針，41ページ，日本規格協会，2000．（対応する国際規格技術報告番号はISO/IEC TR 10176:1998である．）

[清兼 98]　清兼義弘，末廣陽一編著（日本DEC）：国際化プログラミンーI18Nハンドブック，オーム社，480ページ，1998．

A.3.6　プログラミング研究会など学会での報告

プログラム言語の日本語化から国際化への転換時期の 1988 年 5 月に，“ 日本語プログラミング ” の特集テーマで情報処理学会プログラム言語研究会が開催され，次の 12 件の論文が発表された．この中で，[平林 88]，[宇土 88]および[木村明 88]は **A.4** の本格的な日本語プログラム言語に関連する報告であり，[小林 88]と[和田孝 88]は **A.5** の日本語仕様記述言語に関連する報告である．残りの 7 件は本節に関連しており，日本語データ処理を中心に議論している．

[小林 88]　小林知宏，内山明彦（早大）：部品化・再利用向き言語 “ mCLD ” によるプログラミング，情報処理学会プログラミング言語研究会資料（16-1），pp.1-8，1988-5-13.

[平林 88]　平林雅英：C 言語の日本語化とその効果，同上 （16-2），pp.9-16，1988-5-13.

[宇土 88]　宇土正浩，佐貫俊幸（日本 IBM）：日本語 APL，同上（16-3），pp.17-23，1988-5-13.

[木村明 88]　木村明（MSA），片桐明（リギーコーポレーション）：日本語プログラミング言語「Mind」についてその概要と，日本語プログラミングの実用性，同上 （16-4），pp.25-32.

[和田孝 88]　和田孝，土田賢省，杉山高弘，阪田全弘，富田兼一，宮下洋一（日電）：プログラミング自動生成のための日本語仕様記述言語，同上（16-5），pp.33-40，1988-5-13.

[中田 88] 中田育夫（筑波大）：プログラミング言語における日本語化の現状と今後の方向（総論），同上（16-6），pp.41-44，1988-5-13.
[和田英一 88] 和田英一（東大）：プログラミング言語で使う文字コード，同上（16-7），pp.45-48，1988-5-13.
[黒田 88] 黒田幸明（NTT）：FORTRAN における日本語化の現状と今後の方向，同上（16-8），pp. 49-52，1988-5-13
[床分 88] 床分眞一，今城哲二（日立）：COBOL における日本語機能の現状と今後の方向，同上（16-9），pp.53-56，1988-5-13.
[猪瀬 88] 猪瀬武久（日電）：C における日本語化の現状と今後の方向，同上（16-10），pp. 57-60，1988-5-13.
[伊集院 88] 伊集院正（NTT），石畑清（東大）：Ada における日本語化の現状と今後の方向，同上（16-11），pp.61-64，1988-5-13
[元吉 88] 元吉文夫（電総研）：Common Lisp における日本語化，同上（16-12），pp.65-68，1988-5-13

SNOBOL4，CLU，Ada，Smalltalk 及び LISP における日本語処理の研究報告を次に示す．SNOBLE4 は大学のセンタで公開され利用された．

[吉田和幸 85] 吉田和幸（大分大），牛島和夫（九大）：SNOBOL4 既存処理系への日本語テキスト処理機能の追加，コンピュータソフトウェア，Vol.2, No.3, pp.518-528，1985.
[中村 85] 中村昭次，久野靖，木村泉（東工大）：日本語 CLU システムの試作，情報処理学会第 30 回（昭和 60 年前期）全国大会，1R-7，pp.461-462，1985.
[福山 85] 福山峻一，小林吉純，伊集院正（NTT），木下恂，落合正俊（東芝）：Ada 日本語処理機能の実現法，情報処理学会第 30 回（昭和 60 年前期）全国大会，1R-6，pp.459-460，1985.
[松尾 86] 松尾篤弥，牛島和夫（九大）：Ada における日本語テキスト処理パッケージの構築とその使用， 情報処理学会記号処理研究会（36-2）／プログラミング言語研究会（5-2）共催資料，7 ページ，1986-3-10.
[尾崎 86] 尾崎正治，松永義文，上林憲行（富士ゼロックス）：プロトタイプ Smaalltalk-80J～Smaalltalk-80 の日本語化についての検討～，情報処理学会

第 32 回（昭和 61 年前期）全国大会，5F-1，pp.425-426，1986.
[元吉 87]　元吉文夫(電総研)：Common Lisp における日本語処理方式の提案，情報処理学会記号処理研究会資料（40-6），8 ページ，1987-1-13.
[黒川利明 89]　黒川利明（日本 IBM），湯淺太一（豊橋技科大），橋本ユキ子（日電），梅村恭司（NTT），伊藤貴康（東北大）：LISP における国際文字処理方式に関する技術的諸問題，情報処理学会記号処理研究会資料（51-1），pp.1-8 ，1989-6-2.
[安本 92]　安本太一（愛知教育大），湯淺太一（豊橋技科大）：Kyoto Common Lisp の日本語文字処理機能の実現とその評価，コンピュータソフトウェア，Vol.9，No.5，pp.391-402，1992.

A.4　本格的な日本語プログラム言語

本節では，予約語まで日本語化した「本格的な日本語プログラム言語」の文献と商用化事例を紹介する．先ず，1 バイトコードでの分かち書きのカタカナ記述が試みられた．その後，2 バイトの漢字コードの普及にともない，分かち書きの漢字かな混じりの記述が可能となった．近頃は，非分かち書きの研究も行われている．

言語仕様は，C++などの既存プログラム言語の構文を日本語化したものと，新規に文法を定めたものに分かれる．

A.4.1　カタカナ記述

A.4.1.1　日本語 COBOL

1960 年代半ばに，日電が COBOL の識別名のカタカナ化だけでなく，予約語までカタカナにした日本語 COBOL を商品化した．これは，**A.3.1**[COBOL 研 67]に言及されており，最初の事例であろう．ただし，1970 年代のコンピュータでは支援されなかった．このマニュアルは筆者が入社当時（1969 年）に読んだ記憶がある．今回の文献調査で日電の COBOL 担当者に依頼し公表文献を探していただいたが見つからなかった．日本のコンピュータに関する研究・開発の貴重な記録が，学会およびメーカともきちんと保存されていないことは問

題である．

日本語プログラム言語は，一部を除き，コンピュータや OS の世代交代についていけず，稼動プラットフォームも限定されている．このため，利用者はその生産性・保守性の良さを認めながらも，本格的な利用に踏み切れていない．これをどう打破するかが，今後でてくる日本語プログラム言語の大きな課題である．

A.4.1.2 ヤチマタと小朱唇

1970 年代のヤチマタはデータベース照会言語だが, 本格的に日本語に取り組んだ初期の研究として見逃せない．国語学者の水谷の小朱唇は，カタカナの予約語を用いた日本語プログラム言語で，日本語の言語処理を目的としており，東京女子大の教育に使用された．この処理系は黒川が開発した．彼の著書（**A.1** [黒川利明 98]，**A.2.1** [黒川利明 90]）にも簡単な説明がある．ヤチマタと小朱唇の文献を次に示す．

[諸橋 76] 諸橋正幸, 藤崎哲之助, 鷹尾洋一, 間下浩之, 渋谷政昭（日本 IBM）：「ヤチマタ」における擬似日本語，情報処理学会計算言語学(CL)研究会資料（8-2），pp.1-10，1976-12-17.

[藤崎 86] 藤崎哲之助（日本 IBM）：日本語によるデータベース照会，日本語情報処理（高橋延匡（東京農工大）編），pp.205-252，1986.

[水谷 85] 水谷静夫（東京女子大）：国語屋から見た総論， bit 別冊 ワープロと日本語処理（石田晴久（東大），木村泉（東工大），安田寿明（東京電機大）編），共立出版，pp.19-30，1985.

[水谷 86] 水谷静夫：次第書き言語《小朱唇》の設計思想， 情報処理学会記号処理研究会（36-3）／プログラミング言語研究会（5-3）共催資料，pp.1-8，1986-3-10.

A.4.1.3　CORAL と CANO-AID

1970 年代後半にキヤノンが，社内のソフトウェア開発生産性向上のため，カタカナ記述の仕様書言語 CORAL を開発した．この開発に協力した日立が CORAL を“漢字 CORAL”という名前で商用化し，読解性を高めるために仕様書（ソースリスト）を漢字に変換・印字するなどの機能強化をした．キヤノンソフトウェアは，CORAL にパターンと部品の概念を取り入れた CANO-AID を商用化した．これは，日立，IBM，富士通のメインフレームと，UNIX や PC などのオープン環境でも稼動し，現在も機能拡張が続いている．CORAL 関連の文献を次に示す．プログラム例を**図 A.3** と**図 A.4** に示す．

[原田實郎 82]　原田實郎，仁平博三：オンラインデータベース向き業務プログラム開維持支援システム「漢字 CORAL」, 日立評論, Vol.64, No.5, pp.351-354, 1982.
[斎藤 91]　斎藤伸隆（キヤノンソフトウェア）：CANO-AID の開発から商品化まで，ソフトウェア生産工学ハンドブック（技術士ソフトウェア研究会編），pp.850-858，フジ・テクノシステム，1991.
[日経データプロ 94]　製品レポート CANO-AIDⅢ, 日経データプロ・ソフト, NS2-544-301，1994-2.
[日経データプロ 92]　ケース・スタディ　丸紅　HITAC 向け CASE ツールで IBM 用ソフトの開発も可能に, 日経データプロ・ソフト, NS2-501-701, 1992-3.
…CANO-AID の使用事例報告.

```
                ステートメント

  ＃レポート.
＊
＊              レポート
＊
  ＃10.
                カイシシヨリ　オ　シヨリ　スル.
                ＬＯＷ          --->     ＆キーエリアＳ.
  ＃10-10.
                レポートフアイルーヨコヨミ　オ　シヨリ　スル.
    プリント    レポートーシユツリヨク　オ　シヨリ　スル.
                ＆キーエリア　＝　ＨＩＧＨ－ＶＡＬＵＥ　デナケレバ
                                              ＃10-10　ヘ　イク.
                シユウリヨウシヨリ　オ　シヨリ　スル.
```

図 A.3　CORAL で記述したパターンの例　CORAL で記述したレポート作成パターンを示した．このパターンの場合，「初期処理」，「レポート作成用ファイルの読み込み」，「レポート出力処理」，「終了処理」については，パターンがロジックをあらかじめ備えている．ただし，たとえばどのような条件のファイルを読み込むかは，「レポート作成用ファイルの読み込み」に対し，ユーザが固有ロジックを記述した「部品」を組み込む必要がある．　　（[日経データプロ 94]より）

```
                ステートメント

【(ファイル，A2，A3，A4，A5，A6，A7，A8，A9，ブロックＳ）
  ＃レポートファイルーヨコヨミ
＊
＊          ファイル　ヨコヨミ
＊
  ＃10.
＊
              SPACE        --->       インプツトーワーク.
              ZERO         ===>       インプツトーワーク G.
              フアイル　オ　ヨム.
              SPACE        --->       ＆ステータス.
              オワリ　ナラ
                  HIGH-VALUE     --->  ＆キーエリア.
                  デル.
              フアイル         ===>     インプツトーワーク.
              ブロックＳ　オ　シヨリ　スル.
              レポートーワード　オ　ジツコウ.
              ＆ステータス　＝　‘スキップ’　ナラ
                                    ＃10        ヘ　イク.
              SPACE          --->     ＆キーエリア.
              インプツトワーク
                             ===>     ＆キーエリア.
```

図 A.4　CORAL で記述した部品の例　図 A.3 中の「レポート作成用ファイルの読み込み」部分に対するユーザ固有のロジックを記述した．このユーザ固有のロジック部分も，「部品」としてディクショナリで管理する．([日経データプロ 94]より)

A.4.2　漢字カナ混じり記述（分かち書き）

A.4.2.1　日本語 AFL と Mind：先駆的な商用化

日本語 AFL（製品名は和漢）と Mind は，PC 上で商用化された漢字カナ混じりの日本語プログラム言語（分かち書き）として先駆的な製品であり，1980 年代に注目された．Mind は，**A.1** [玉井 98]でも言及されており，16 ビット版の製品は現在も市販されている．この言語は Forth 言語のスタックの考え方を基調としており，テレビ放送でも紹介された．適用事例も多く，富士通が自社 PC を拡販するためディーラ受注の業務プログラム記述言語として推奨し利用された．中高等学校の教育用プログラム言語としても教育界で評価検討され授業でも使われた．日本語 AFL と Mind の文献を次に示す．プログラム例を**図 A.5，図 A,6** および**図 A.7** に示す．

[菅野 83]　菅野淳，本田邦夫，岡村嘉巳，上田謙一（松下技研）：自然言語の利用でマン・マシーン・インターフェースを改善する日本語 AFL，情報処理学会コンピュータにおけるヒューマン・インターフェースシンポジウム報告集，pp.53-65，1983-7-22.

[鈴木孝則 83]　鈴木孝則（国際データ機器）：日本語プログラミング言語『和漢』，情報処理学会マイクロコンピュータ研究会資料（29-2），10 ページ，1983-11-28.

[菅野 85]　菅野淳，上田謙一（松下技研）：日本語 AFL，bit 別冊 ワープロと日本語処理（石田晴久，木村泉，安田寿明編），pp.243-251，1985.

[那野 83]　那野比古：要説日本語 AFL，p.189，東京ブック，1983.

[片桐 84]　片桐明（リギーコーポレーション）：FORTH，パソコン言語学（石田晴久監修），pp.255-288，1984.

[片桐 85]　片桐明(リギーコーポレーション):日本語プログラミング FORTH, bit 別冊 ワープロと日本語処理(石田晴久, 木村泉, 安田寿明編), pp.236-242, 1985.

[平賀 89]　平賀正樹（富士通），日本語プログラム言語 Mind，コンピュータソフトウェア，Vol.6，No.2，pp.170-178，1989.

[片桐 84]　片桐明（リギーコーポレーション）：FORTH，パソコン言語学（石

田晴久監修)，pp.255-288，1984.
[片桐 95a]　片桐明：日本語プログラム言語 Mind 基本文法（第３刷)，287 ページ，リギーコーポレーション，1995-6.
[片桐 95b]　片桐明：日本語プログラム言語 Mind PureMindPRO5.2 ユーザズマニュアル（第７版３)，248 ページ，リギーコーポレーション，1995-11.
[NHK88]　NHK 取材班編：日本語プログラミングの試み，“ 世界の中の日本：コンピュータが日本を変える　第２巻（コンピュータは言葉の壁を越えられるか)”，pp.210-223，日本放送協会／角川書店，1988.
[加野島 89]　加野島英雄，大網呼人：日本語でプログラムを書く方法，143 ページ，翔泳社，1989.
[西之園 93] 西之園春夫（研究代表者：京都教育大学)：情報教育のための日本語プログラミング言語 Mind の実用化に関する研究，平成 4 年度科学研究費補助金（試験研究（B)（1))研究成果報告書（課題番号：03558030)，1993, 約 400 ページ.

```
1　A組　は　上田　160　70
2　　　　佐藤　170　80
3　　　　……
4　肥満度　は　[体重/((身長－100)＊0.9)＊100],
5　A組　の　行　の　数　を　取り出し，N　とする.
6　K　は　1,
7　新A組　は　[　].
8　[A組　から　K　行目　を　取り出し，S　とする.
9　　S　から　1　項目目　を　取り出し，　名前　とする.
10　　S　から　2　項目目　を　取り出し，　身長　とする.
11　　S　から　3　項目目　を　取り出し，　体重　とする.
12　　肥満度を計算する.
13　　S　と　[　]　と　結果　を　照合し，　S　とする.
14　　新A組　の　K　行目　に　S　を　入れる.
15　　K　に　1　を　加える.]　を
16　N　回　繰り返す.
```

図 A.5　日本語 AFL のプログラム例　（[岩波辞典 90]より）

```
：階乗　これが　１より　大きい　ならば
        これから　１を　引き
        階乗　を取り　掛ける　そして;
```

図 A.6　Mind のプログラム例：階乗計算　（[岩波辞典 90]より）

```
検索ファイルは　ファイル。
検索文字列は　　文字列実体。

検索処理とは　　（・　――＞　・）
        文字列１は　文字列
    ここから
        検索ファイルから　一行読み出し　文字列１に　入れ
        データ終り？
            ならば　打ち切り
            つぎに
        文字列１から　検索文字列を　検索したものが　真？
            ならば　文字列１を　表示し　改行する
            つぎに
    繰り返すこと。

エラー処理とは　　（・　――＞　・）
    エラー？
        ならば　エラー文字列で　重大エラーする
        つぎに。

メインとは
    ブレーク抑止解除し　　(CTRL-C 打鍵によるブレークを有効にする)
    起動パラメータから　単語切り出しし　検索ファイルを　オープンし
    エラー処理し
    起動パラメータから　単語切り出しし　検索文字列に　入れ
    検索処理し
    検索ファイルを　クローズすること。

＊　終り　＊
```

図 A.7　Mind のプログラム例：文字列検索　（[片桐 95a]より）

A.4.2.2　第 4 世代言語での日本語プログラミング

大手メーカーの基幹業務向けの開発支援システムや第４世代言語では，ロジックの記述に日本語を用いているものが多い．それぞれ，その時代時代の日本語環境と整合のとれた実用的な日本語プログラム仕様を実現し，生産性を上げ

る努力をしている．日立の EAGLE/4GL（**図 A.8**）や富士通の YPS（**図 A.9**）は，出荷数が万を超えている．既存の英数字の COBOL プログラムを入力し，保守用ドキュメントとして識別名などを漢字に変換・印字するツールも利用されており，日立の ADCAS などはその商用化例である．これらの製品の文献を次に示す．

[今城 89] 今城哲二，秋山美登（日立），北尾修二，里本健，脇坂隆則（日立ソフトウェアエンジニアリング），大西俊治（日立）：VOS K 第４世代言語“EAGLE/4GL”，日立評論，Vol.71，No.11，pp.1119-1124，1989.

[今城 90a] Tetsuji Imajo, Yoshinori Akiyama（日立）and Shuji Kitao（日立ソフトウェアエンジニアリング）：VOS K 4th Generation Language “EAGLE/4GL”， HITACHI REVIEW, Vol.39, No.5, pp.261-266, 1990.

[西尾 93] 西尾高典，秋山美登，今城哲二(日立)：アプリケーションの分散開発を実現する第 4 世代言語，日立評論，Vol.75，No.9，pp.605-610，1993-9.

[吉野 93] 吉野松樹，田村和敏（日立），稲益良夫（日立ソフトウェアエンジニアリング）：ソフトウェア開発支援ツール“SEWB3，EAGLE/4GL”の機能と特長，日立評論，Vol.75，No.11，pp.727-734，1993-11.

[日立 95] 日立製作所：SEWB3 第 4 世代言語文法マニュアル（第 3 版），3000-7-475-20，533 ページ，1995.

[平川 86] 平川知親，中村康一，谷部幸男（日立），戸金旗一（日立コンピュータコンサルタント）：自動ドキュメンテーション支援システム“ADCAS”－金融アプリケーションパッケージへの適用例－，日立評論，Vol.68，No.5，pp.369-372，1986.

[村上 87] 村上憲稔，宮成功，富田嘉文，野沢隆，林義雄（富士通）：YACⅡエディタ，構造エディタ，pp.135-146，共立出版，1987.

[植木 88] 植木定裕，宮内和人，中田晴美（富士通）：高生産ツール CASET，FUJITSU，Vol.39，No.1，pp.21-28，1988.

[三上 88] 三上次郎, 東村昭男(富士通)：高生産ツール YPS, FUJITSU, Vol.39, No.1，pp.29-35，1988.

[阪井 90] 阪井隆晴，楢府坦，今津幸雄（富士通）：アプリケーション開発支援ツールとディクショナリシステム，FUJITSU，Vol.41，No.5，pp.403-412，

1990.
[日経データプロ 97a] 製品レポート CASET，日経データプロ・ソフト，NS2-544-701，1997-3.
[日経データプロ 97b] 製品レポート YPS 日経データプロ・ソフト，NS2-544-721，1997-3.

```
************************************************************
  手続き
************************************************************
*
  出力時初期処理
      初期化  S2
*
  入力チェック
      評価  ＰＦキー
          場合  ‘PF02’
          場合  ‘PF03’
          場合  ‘ENTER’
              無処理
          場合    他
              エラー  ‘ＰＦキーエラー’
    評価終了
*
  入力データ編集
    判定     ＰＦキー  =  ‘ENTER’
        転記    R1.商品コード  <--  S2.商品コード
        読み込み  R1
        判定  @入力不正(R1)
            エラー  ER01  AT  S2.商品コード  DFC  ‘ER’
        判定終了
*
        転記    R2.商品コード  <--  S2.商品コード
        読み込み  R2
        判定  @入力不正(R2)
            エラー  ER02  AT  S2.商品コード  DFC  ‘ER’
        判定終了
*
        転記    WK-商品コード      <--    S2.商品コード
        転記    WK-商品名          <--    R2.商品名
        転記    WK-色              <--    R2.色
        転記    WK-現在庫量        <--    R1.現在庫量
        転記    WK-最高在庫量      <--    R1.最高在庫量
        転記    WK-安全在庫量      <--    R1.安全在庫量
    判定終了
```

図 A.8　EAGLE/4GL のプログラム例　　([日立 95]より)

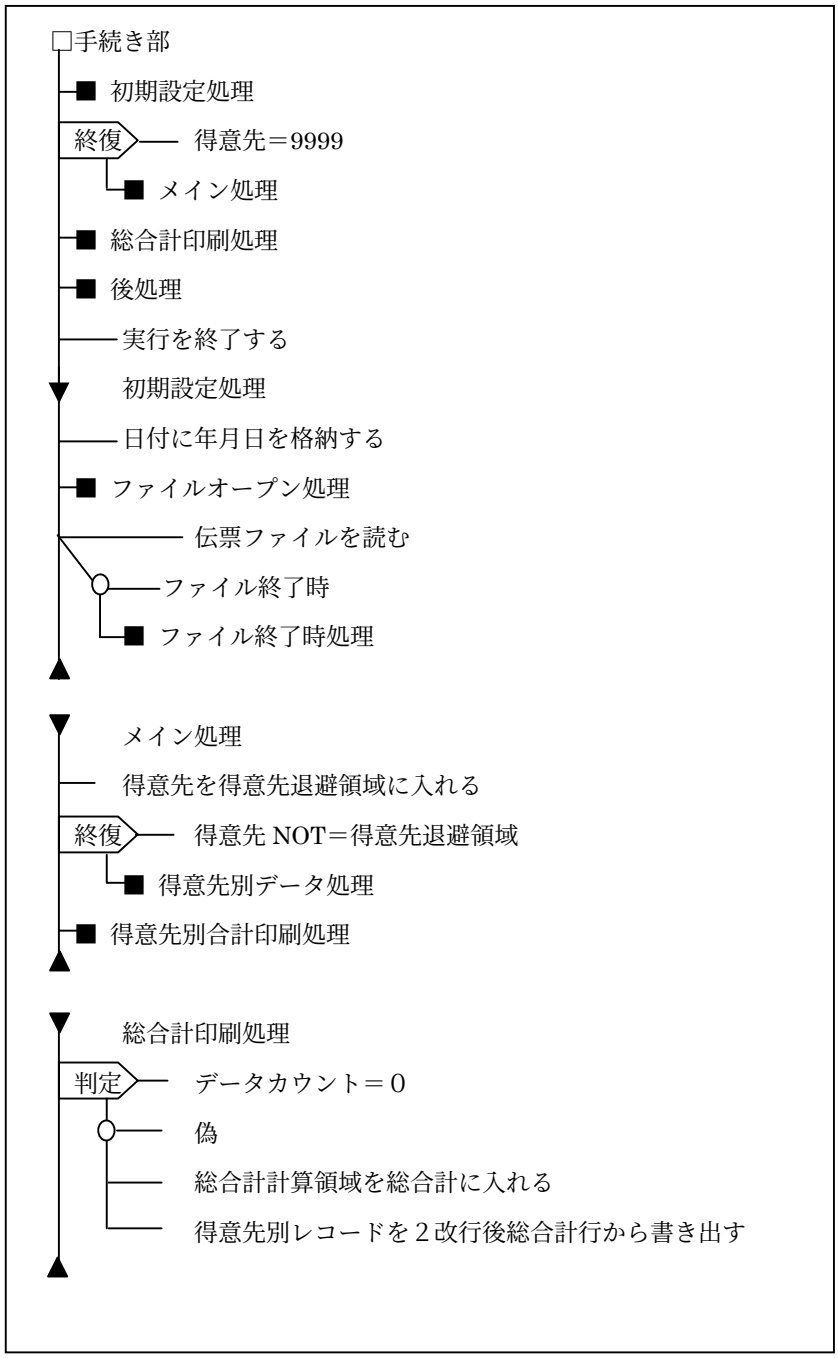


図 A.9　YPS のプログラム例　富士通の日本語プログラミング表記法 YACⅡを用いてロジックを記述する．　（[日経データプロ 97b]より）

A.4.2.3　学校教育用オブジェクト指向言語“ドリトル”

2002 年から中学で，2003 年から高校で情報教育が必修となる．この学習指導要領では，“ プログラム言語によるプログラミングは取り上げなくてもよい ” となっている．これについいては識者から問題視されているが，教育現場で教えるプログラム言語として適切なものがないということも一つの理由で大勢を変えるに至っていない．

次の論文で兼宗などが提案している学校教育用オブジェクト指向言語 “ ドリトル ”は初等教育まで視野に入れて，変数や命令語などに日本語を用いている．処理系は Java2 で書かれたインタプリタとして実装されており，実際の教育現場でも実験が開始されている．この研究は情報処理振興協会（IPA）の 2000 年度未踏ソフトウェア創造事業の補助を受けており，新聞や雑誌でも報道された．これがより成長し，教育現場で広く使われることを期待したい．**図 A.10** にプログラム例を示す．

[兼宗 2000]　兼宗進，久野靖（筑波大）：学校教育用オブジェクト指向言語/環境の構想について，情報処理学会シンポジウムシリーズ（情報教育シンポジウム論文集），Vol.2000，No.9，pp.79-82，情報処理学会，2000-7-29～31.

[兼宗 2001a]　兼宗進，久野靖（筑波大）：学校教育用オブジェクト指向言語 “ Dolittle ” の提案，第 42 回プログラムシンポジウム，pp.16-20，情報処理学会，2001.

[兼宗 2001b]　兼宗進（筑波大），御手洗理英（筑波大／アーマット），中谷多哉子（エス・ラグーン），福井眞吾，久野靖（筑波大）：学校教育用オブジェクト指向言語「ドリトル」の設計と実装，情報処理学会論文誌，Vol.42，No.SIG11(PRO12)，pp.78-90，2001.

[中谷 2001]　中谷多哉子（エス・ラグーン），兼宗進（筑波大），御手洗理英（アーマット），福井眞吾，久野靖（筑波大）：オブジェクトストーム：プログラミング言語と情報教育の新しい関係，オブジェクト指向最前線 2001（情報処理学会 oo2001 シンポジウム），pp.57-64，近代科学社，2001 年 9 月．

```
カメ太＝タートル！作る。
矩形＝カメ太！５０上へ１００右へ５０下へ閉じる図形にする。
矩形！(青)塗る。
白矩形＝矩形！複製５０右移動(白)塗る。
赤矩形＝白矩形！複製５０右移動(赤)塗る。
```

図 A.10　ドリトルのプログラム例　　フランス国旗を描いている．

A.4.2.4　学会などで報告された研究

近頃の研究事例を次に示す．[中鉢 97]は Mind 同様，Forth のスタックの概念を取り入れている．[鈴木康彦 97]は Java を予約語まで含めて日本語化した野口研究室（神奈川大）の研究報告である．野口研究室では同じ考えで Java の中国語化も研究している．大澤の三つの論文は OS テストやシミュレーションンなど特定分野向け言語の日本語化の研究である．[大澤 99]では，分かち書きの空白の読みづらさの解消のために，変数名を斜体やゴシックにするなど字体を変えることや，色や下線を使うことなどを提案している．

[中鉢 97]　中鉢欣秀，大岩元（慶大）：Java ヴァーチャルマシンをターゲットとした日本語オブジェクト指向言語の開発，情報処理学会プログラミング研究会（13-6），pp.31-38，1997-5-21.
[中鉢 98]　中鉢欣秀（慶大）：日本語思考のプログラミング，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.7-9，1999.
[鈴木弘 99]　鈴木弘（都立航空工業高専／慶大），中鉢欣秀，大岩元（慶大）：日本人のための初心者向けプログラミング教育用言語の試作，情報処理学会第 59 回（平成 11 年前期）全国大会，2X-06，1999.
[鈴木康彦 99]　鈴木康彦，野口健一郎，後藤英一（神奈川大）：Java をターゲットにした自国語プログラミングの実験，情報処理学会第 58 回（平成 11 年前期）全国大会，5M-08，1999.
[大澤 93]　大澤範高（パーソナルメディア），木村憲雄（東大）：プログラム言語「きなり」，トロン技術研究会，Vol.5，No.2，pp.39-50，1993-3.

[大澤 95] 大澤範高，弓場敏嗣（電通大）：並列離散事象シミュレーション言語「もえぎ」の構想，信学技報（COMP94-92），pp.25-32，1995-3.
[大澤 99] 大澤範高（文部省メディア教育開発センター）：日本語風表記でプログラミングできる言語「もえぎ」とその処理系，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日， pp.1-6，1999.

A.4.3 非分かち書きの漢字カナ混じり記述

A.4.3.1 日本語 C++

分かち書きは小学校低学年で使われる書き方で，通常の日本語では非分かち書きが使われている．非分かち書きの日本語プログラム言語の代表例が，宮脇の日本語 C++であり，C++の言語仕様と対応した構文をすべて備えている．処理系も日本語 C++で記述しており，非常に読みやすい．新しい仕様のプログラム言語を普及させる困難さを考えると，普及済みの既存言語の全面日本語化は一つの見識である．同じ考えの“日本語 COBOL”や“日本語 Java”も有効であろう．日本語 C++に関する論文を次に示す．**図 A.11** にプログラム例を示す．

[宮脇 94] 宮脇富士夫（姫路工大），尾関哲（神戸高専），太田健一，佐藤邦弘（姫路工大）：日本語プログラミング言語（日本語 C++）の開発，姫路工業大学工学部研究報告，No.47，pp.70-82，1994.
[宮脇 93] 宮脇富士夫，佐藤邦弘：日本語プログラミング環境のための開発と評価，「利用者指向の情報システム」シンポジウム，pp.125-134，1993-11.
[尾関 95] 尾関哲，佐藤邦弘，太田健一，宮脇富士夫：プログラム理解における日本語使用の効果，情報処理学会第 51 回（平成 7 年後期）全国大会，7M -3，pp.5-169～5-170，1995.
[尾関 94] 尾関哲，佐藤邦弘，太田健一，宮脇富士夫：日本語プログラミング用エディタ評価のための識別子ベンチマークデータ，情報処理学会論文誌，Vol.35，No.9，pp.1949-1954，1994.
[宮脇 99] 宮脇富士夫，尾関哲，太田健一，佐藤邦弘：日本語プログラミング環境の開発例，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.33-40，1999.

[尾関 99] 尾関哲，宮脇富士夫，太田健一，佐藤邦弘：日本語ログラミング用エディタの開発，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.41-44，1999.

```
10  :              2次方程式の解
20  挿入  <stdio.h>
30  挿入  <math.h>
40  列挙{実数解，複素数解，1実数解，無効} 根。
50  :-------------------------------------------------------
60  空値  2次方程式の解（実数  係数a，実数  係数b，実数  係数c，
70                          実数  ＊根1，実数  ＊根2）
80 {
90  実数  根，判別式。
100  ((係数a＝＝0）＆＆（係数b＝＝0））ならば
110         「根＝無効。呼出点へ戻る。」
120  （係数a＝＝0）ならば
130         「根＝1実数解。＊根1＝－係数c／係数b。呼出点へ戻る。」
140  判別式＝係数b＊係数b－4＊係数a＊係数c。
150  （判別式>＝0）ならば
160         「根x＝（絶対値（係数b）＋  平方根（判別式））／（2＊係数a)。
170         （係数b＞0）ならば  根x＝－根x。
180           ＊根1＝根x。
190         （根x！＝0）  ならば  ＊根2＝（係数c／係数a）／根x。
200         そうでなければ ＊根2＝0。
210         根＝実数解。」
220  そうでなければ
230         「＊根1＝－係数b／（2＊係数a)。
240           ＊根2＝平方根（－判別式）／絶対値（2＊係数a)。
250           根＝複素数解。」
260 }
270  :-------------------------------------------------------
280  空値  主プログラム（  ）
290  {
300  実数  係数a，係数b，係数c，根x，根y。
310  画面表示（“2次方程式の根を求める。￥n￥n”)。
320  繰返条件（。キー入力（“%g%g%g，＆係数a，＆係数b，＆係数c）！＝キー入力終了。）
330               で次の文を繰り返す。
340  「画面表示（“￥na＝%g￥tb＝%g￥tc＝%gt”，係数a，係数b，係数c)。
350    2次方程式の解（係数a，係数b，係数c，＆根x，＆根y)。
360    （  根）が
370    1「実数解  の場合  画面表示（“根＝%g，%g￥n”，根x，根y)。繰返終了。
380      複素数解  の場合  画面表示（“根＝%g，＋－%g  i￥n”，根x，根y)。繰返終了。
390      1実数解  の場合  画面表示（“根＝%g，無限大￥n”，根x)。繰返終了。
400      無効      の場合  画面表示（“a＝b＝0？￥n”)。繰返終了。」1」
410 }
```

図 A.11　日本語 C++のプログラム例　（[宮脇 94]より）

A.4.3.2　日本語プログラム言語“まほろば”

A.2.1 で紹介した[今城 99b]では，非分かち書き日本語プログラム言語の言語仕様設計時の留意点について論じている．その設計指針に基づき設計された日本語プログラム言語“まほろば”は，PASCAL，C および COBOL を参考にした言語で，通常のアルゴリズムとデータ構造の記述と事務処理を目的としており，次の文献で報告されている．これらは本論文の 4. と付録 B の母体である．

[今城 2001a]　今城哲二（日立／奈良先端大），鈴木弘（都立航空工業高専），大野治（日立），植村俊亮（奈良先端大）：日本語プログラム言語“まほろば”の言語仕様，情報処理学会ソフトウェア工学研究会研究報告，2000-SE-130，pp.143-152，2001 年 3 月．
[今城 2000]　今城哲二（日立）：第 4 章　言語処理系，入門計算機ソフトウェア（金子敬一（東京農工大），今城哲二（日立），中村英夫（日大）共著），pp.76-130，朝倉書店，2000 年 4 月．

A.4.4　学会での発表など

本項では，学会などで発表された日本語プログラム言語関連の文献の中で，記録に留めておきたいものを列挙する．日本語 APL は日本 IBM により商用化された．予約語も日本語化した BASIC の例を図 A.12 に示す．[小谷 93]の助詞の使用法の考察は，日本語プログラム言語を新たに設計するときに示唆に富む．[芹沢 94]は商用の日本語 Logo の解説である．

[島崎 81]　島崎真昭（京大）：日本語 Pascal：パスカル，情報処理学会第 23 回（昭和 56 年後期）全国大会，1H-1，pp.215-216，1981.
[森本 84]　森本陽二郎，中山康子，南川忠利（東芝）：パソコン用オブジェクト指向型言語，情報処理学会第 29 回（昭和 59 年後期）全国大会，2L-1，pp.1549-1550，1984.
[平尾 86]　平尾隆行（日本 IBM）：日本語 APL 入門，222 ページ，オーム社，

1986.
[紺屋 87] 紺屋真一（富士通）：BASIC 言語における国語機能のサポート，情報処理学会第 32 回（昭和 62 年前期）全国大会，3V-2，pp.827-828，1987.
[木村俊一 87] 木村俊一（長野信連）：日本語プログラミング，昭和 62 年度人口知能全国大会（第 1 回）9-5，pp.465-468，1987.
[小谷 87] 小谷善行，末永富美代，高田正之（東京農工大）：日本語 Prolog の利用者インターフェース，情報処理学会第 34 回（昭和 62 年前期）全国大会，3U-1，pp.743-744，1987.
[渡辺坦 89] 渡辺坦（日立）：パタン照合型プログラミング方式の提案，情報処理学会プログラム言語研究会報告（20-1），pp.1-9，1989-2-10.
[小谷 93] 小谷善行（東京農工大）：日本語に基づく論理プログラム表現，情報処理学会論文誌，Vol.34，No.5，pp.973-984，1993.
[吉田正和 93] 吉田正和：日本語 C プログラミング，インターフェース，1993 年 4 月号，pp.242-244，CQ 出版社.
[芹沢 94] 芹沢浩（都立青鳥養護学校）：日本語 Logo 入門，183 ページ，森北出版，1994.
[榊原 96] 榊原毅，乾伸雄，野瀬隆，小谷善行，西村恕彦（東京農工大）：日本語的な文法をもつ LOGO の文法設計，情報処理学会第 52 回（平成 8 年前期）全国大会，2N-1，pp.5-1～5-2，1996.

日本語化した BASIC	従来の BASIC
100　画面消去：出力“同心円を描きます”	100　CLS:PRINT “同心円を描きます”
110　＊座標入力	110　＊GETZAHYOU
120　出力“中心座標を指定してください”	120　PRINT“中心座標を指定してください”
130　入力“Ｘ座標”; 座標Ｘ	130　INPUT “Ｘ座標”; ZAHYOUX
140　入力“Ｙ座標”; 座標Ｙ	140　INPUT “Ｙ座標”; ZAHYOUY
150　判定 座標Ｘ＜０ 論理輪 座標Ｙ＜０ 成立時＊座標入力	150　IF ZAHYOUX＜０ OR ZAHYOUY＜０ THEN *GETZAHYOU
160　反復　半径＝10 から 100　増分　10	160　FOR HANKEI＝10 TO 100 STEP 10
170　円（座標Ｘ，座標Ｙ），半径	170　CIRCLE (ZAHYOUX, ZAHYOUY), HANKEI
180　繰り返し	180　NEXT

図 A.12　BASIC の予約語と識別名の日本語化　[紺屋 87]より）

A.5　日本語仕様記述言語

A.5.1　企業や大学の研究

企業や大学での日本語による仕様書記述言語の研究の文献を次に示す. この研究は 1980 年代を中心に広く実施され多くの報告が発表された．中でも沖電気／九大の NBSG に関する文献が多い. 大部分は研究段階で終わっており，一部は社内ツールとして利用されたが，本格的な実用には至らなかった．近頃のこの分野の研究は，ビジュアル化と自然語からの仕様抽出が主要テーマとなっており，単なる日本語による仕様書記述の研究報告は少ない. 1987 年までの仕様書記述言語の文献は，[大野 89]が詳しい.

[南 83]　南俊郎（九大），杉尾俊之，武内惇，椎野努（沖電気）：日本語をベースとした仕様記述言語 NBSG，情報処理学会ソフトウェア工学研究会資料（30-A），7 ページ，1983-6-29.

[杉尾 84]　杉尾俊之，武内惇，椎野努（沖電気）：日本語をベースにした仕様記述言語 NBSG における仕様記述について, 情報処理学会ソフトウェア工学研究会資料（34-13），pp.73-84，1984-2-10.

[山之上 87]　山之上卓，安在弘幸，吉田将（九大），杉尾俊之，武内惇，椎野努（沖電気）：言語処理系の生成系 MYLANG による NSBG/PD プリコンパイラの試作，情報処理学会論文誌，Vol.28，No.1，pp.64-73，1987.

[上原三八 83]　上原三八，佐藤秀樹，毛利友治，小野越夫，高尾哲康（富士通研）：詳細設計支援システム PDAS におけるプログラム生成，情報処理学会第 27 回（昭和 58 年後期）全国大会，3C-3，pp.497-498，1983.

[杉山 84]　杉山健司，秋山幸司，亀田雅之，竹之内顕文（富士通研），対話型自然言語プログラミングシステムの試作，電気通信学会論文誌，Vol.J67-D，No.3，pp.297-304，1984.

[稲田 86]　稲田満，岡本務，渡辺敏，中村雄三（NTT）：プログラム設計用言語 SL の開発と評価, 情報処理学会論文誌, Vol.27, No.6, pp.601-611, 1986.

[上原邦昭 86]　上原邦昭（阪大），藤井邦和（日本 IBM），豊田順一（阪大）：自然言語による自動プログラム合成, コンピュータソフトウェア, Vol.3, No.4, pp.359-368，1986.

[大西 87] 大西淳，阿草清滋，大野豊（京大）：要求定義のための要求フレーム，情報処理学会論文誌，Vol.28，No.4，pp.367-375，1987.
[大西 90] 大西淳（京大），阿草清滋（名大），大野豊（京都高度技術研究所）：要求フレームに基づいたソフトウェア要求仕様化技法，情報処理学会論文誌，Vol.31，No.2，pp.175-181，1990.
[大野豊 89] 大野豊（甲子園大），原田実（青山学院大）：自動プログラミングハンドブック，457 ページ，オーム社，1989.
[岩元 87] 岩元莞二，西谷泰昭，和田孝（日電）：プログラム自動生成システム，NEC 技報，Vol.40，No.1，pp.35-38，1987.
[千吉良 87] 千吉良英毅，永松祐嗣，小林正和（日立）：システム仕様書の再利用によるソフトウェアの開発技法(ICAS-REUSE)，日立評論，Vol.69，No.3，pp.249-254，1987.
[上原憲二 88] 上原憲二，黒田清隆，土井日輝，行徳孝彦，鈴木由美子（三菱電機）：プログラム自動生成システム SAGE，情報処理学会ソフトウェア工学研究会資料（58-10），pp.73-79，1988-2-4.
[大石 88] 大石東作（電総研）：ソフトウェアの「日本語仕様書」作成支援，情報処理学会ソフトウェア工学研究会資料（59-4），7 ページ，1988-5-18.
[原田実 93] 原田実，中村義幸（青山学院大）：プログラムの構造と論理の自動設計システム EOS/M，情報処理学会論文誌，Vol.34，No.9，pp.2013-2024，1993.
[松野下 96] 松野下博司，園田雅紀，河合敦夫，椎野努（三重大），武内惇（日大）：仕様書記述制限日本語に対する記述支援系の開発，情報処理学会第 53 回（平成 8 年後期）全国大会，2D-6，pp.1-219～1-220，1996.

次の文献で紹介されている加藤木と畠山などの一連の研究は，仕様書記述言語の分野で現在も研究が行われている貴重なものである．母国語の日本語をベースとし，オブジェクト指向分析・設計・プログラミングの各工程を対象とした日本語一貫プログラミング環境を提案しており，システム開発で実用化されている．

[加藤木 98] 加藤木和夫（日立プロセスコンピュータエンジニアリング），畠

山正行（茨城大）：オブジェクト指向日本語一貫プログラミング環境，情報処理学会第 118 回ソフトウェア工学研究会報告，98-SE-118， pp.15-22，1998.
[加藤木 99] 加藤木和夫（日立プロセスコンピュータエンジニアリング），畠山正行（茨城大）：オブジェクト指向日本語一貫プログラミング環境，情報処理学会論文誌，Vol.40，No.7，pp.3016-3030，1999.
[石井 99] 石井義之，畠山正行（茨城大），加藤木和夫（日立プロセスコンピュータエンジニアリング）：オブジェクト指向日本語分析記述環境 OOJ の設計開発，情報処理学会第 122 回ソフトウェア工学研究会報告，99-SE-122，pp.55-62，1999.
[畠山 2000] 畠山正行（茨城大），加藤木和夫（日立プロセスコンピュータエンジニアリング），石井義之（茨城大）：オブジェクト指向記述日本語 OODJ とその記述環境，情報処理学会論文誌，Vol.41，No.9，pp.2567-2581，2000.
[上田 2002] 上田賀一，畠山正行（茨城大），加藤木和夫（日立情報制御システム）：オブジェクト指向計算記述日本語 OCDJ の設計，信学技報（電子情報通信学会技術研究報告［ソフトウェアサイエンス］），Vol.101, No.629, pp.39-46, 2002.

A.5.2 アルゴリズム記述用の日本語擬似言語

土居と筧が編集した岩波の入門シリーズでは，アルゴリズムを日本語擬似言語で記述しており，分岐，繰り返し及び文の先頭に三角，四角，丸（▲（を時計回りに 90 度回転した記号），■，●）の記号を用いて，視覚性を高め，フローチャートの役割も果たしている．ただし，処理系はない．**図 A.13** に記述例を示す．

1999 年代後半から，大学入試センター試験の数学の中に“ 情報関連基礎 ”が新設された．そこでは当初は COBOL，C および BASIC のプログラム言語の問題が出されていたが，1998 年からはアルゴリズムを問う問題に変わり，日本語擬似言語でアルゴリズムが記述されるようになった（**図 A.14**）．第二種情報処理技術者（2000 年度から基本情報技術者）試験でも 1999 年秋期からアルゴリズムを問う問題に，土井・筧の記法に似た記号を用いた日本語擬似言語が使われている．意識的かどうか分からないが，いずれの試験もその表記法は毎回

アルゴリズム 2.11　除算法

変数　$N, x, y, b, s, gcd, lcm,$：正の整数

●入力データの組数を N に入力する

■ $N > 0$ である限り

　●二つの整数を変数 x と y に入力する

　▶ $x > 0$ でしかも $y = 0$ ならば

　　● $b \leftarrow x$

　　● $s \leftarrow$ y

　　■ $s \neq 0$ である限り

　　　● r $\leftarrow b$ mod s

　　　● $b \leftarrow s$

　　　● $s \leftarrow$ r

　　● $gcd \leftarrow b$

　　● $lcm \leftarrow \dfrac{x \times y}{gcd}$

　　● x, y, gcd, lcm の値を出力する

　●$N \leftarrow N - 1$

図.A13　土居・箕記法による日本語擬似言語のアルゴリズム記述例

除算法により最大公約数と最小公倍数を求めている．（[土居 87] より）

```
a←1，b←1，k←1
a ≦ NA［ア］b ≦ NB である間
|   もし［イ］＜［ウ］が成立すれば
|   |       SinY[k]←SinA[a]，BanY[k]←BanA[a]，a←a＋1
|   成立しなければ
|   |       SinY[k]←SinB[b]，BanY[k]←BanB[a]，b←b＋1
|   を実行する
|   k←k＋1
を繰り返す
```

図 A.14　大学入試センター試験での日本語擬似言語のアルゴリズム記述

（1999 年数学「情報関連基礎」問 1 より）

変わっている．試験問題という性格により，公の場でこれら擬似言語の仕様の議論ができないのは残念である．

[土居 87]　土居範久（慶大），筧捷彦（早大）：プログラミングの考え方，171 ページ，岩波書店，1987.
筧 89]　筧捷彦（早大）：Pascal と C への招待，223 ページ，岩波書店，1989.
[筧 87]　筧捷彦（早大）：基本プログラミング，244 ページ，岩波書店，1987.
[土居 91]　土居範久：基礎 C 言語，351 ページ，岩波書店，1991.

A.6　まとめ

日本語プログラム言語は, 既存言語の識別名でのカタカナ採用からはじまり, 1980 年以降の漢字コードの普及にともない, 漢字で識別名が書けるようになった．日本語データ処理も当たり前になり，1990 年代には標準プログラム言語の国際規格でも，日本語処理を一般化した国際化機能が次々と採用されている．日本語プログラム言語の研究は仕様書言語を含め, 1980 年代に盛り上がりを見せ，予約語まで日本語化した本格的な日本語プログラム言語（分かち書きレベル)が商用化されている. 1990 年代に入ってこの研究･開発は下火になったが，一部では引き続き行われ，1998 年ごろから若干活気がもどってきており，非分かち書きの研究も進んできた．

本付録では，今までの研究と商用化の文献を網羅的に紹介した．このまとめとして，**表 A.1** と**表 A.2** に日本語プログラム言語の発展段階と代表的な日本語プログラム言語の一覧を示す．

システム開発・維持・発展作業が使われる現場（対象世界）の「ことば」は，母語である．プログラミングでもこの母語を利用すると，効果は顕著であるのに，日本で行われているプログラム作成は相変わらず 1 バイト系の英数字を用いて行っているのが大半である．日本語プログラム言語の研究の進歩と普及により，多くの人々が母語である日本語でプログラムを書き，それを多くの人が読んで理解できるようになることを期待したい．

表 A.1　日本語プログラム言語の発展段階

段階	時期	日本語化のレベル
0	1960 年代	予約語・識別名とも英語ベース
1	1960 年代	ローマ字の日本語識別名
2	1960 年代後半	識別名にカタカナを使用
3	1960 年代後半	予約語と識別名をカタカナ化
4	1980 年代前半	識別名に日本語（漢字）を使用
5	1980 年代後半	予約語も漢字を使用．語順は英語流
6	1980 年代後半	予約語・識別名に漢字を使用． 語順は日本語流（書き方は分かち書き）
7	1990 年代後半	同上．（書き方は非分かち書き）

表 A.2　代表的な日本語プログラム語

項番	言語名	文献	表 A.1 の段階
1	COBOL（カタカナ識別名）	[COBOL 研 67]	2
2	日本語 COBOL（日電）	[COBOL 研 67]	3
3	小朱唇	[水谷 86]	3
4	CORAL（後継は CANO-AID）	[斎藤 91]	3
5	国際化機能を支援した COBOL	[今城 99c]	4
6	日本語 AFL	[菅野 85]	6
7	Mind	[片桐 85]	6
8	EAGLE/4GL（第 4 世代言語）	[今城 89]	5
9	YPS	[三上 88]	6
10	オブジェクト指向日本語一貫 プログラミング環境（OODJ）	[加藤木 98] [畠山 2000]	6
11	ドリトル（学校教育用 OO 言語）	[兼宗 2001a]	6
12	日本語 C++	[宮脇 94]	7
13	まほろば	[今城 2001b]	7

付録 B　まほろば 0 言語の文法と処理系 ——プログラム言語“まほろば”の記述実験——

B.1　はじめに

まほろばの言語仕様がアルゴリズム記述に向いているかどうかの評価については **4.**で述べた．ここでは，その評価実験用に作成した“まほろば 0”言語の文法とその処理系について説明し，処理系のソースリストを示す．

“まほろば 0”の文法はまほろばのサブセットであり，まほろば 0 処理系はコンパイラとインタプリタとで構成される．いずれも Wirth の PL/0 言語と中田の PL/0’言語の文法と処理系を参考にした[Wirth79]，[中田 96]．コンパイラは 1 パス・コンパイラで，“まほろば 0 仮想計算機”の機械語の目的プログラムを生成後，中間言語をインタプリタで実行する．

本章の構成は次のとおりである．**B.2** でまほろば 0 言語の文法を定義する．**B.3** でその言語のコンパイラの出力（目的プログラム）が動作する仮想計算機と，コンパイラ内部の中間語および目的コードについて述べる．**B.4** でまほろば 0 言語処理系の構成を説明し，**B.5** でソースリストを示す．**B.6** でまとめを述べる．

B.2　まほろば 0 言語の文法

まほろば 0 言語のデータ型や定数の種類や文の種類は，Wirth の PL/0 言語および中田の PL/0’言語とほぼ同等な記述能力を持つようにしたが, 関数（PL/0 ではプログラム）のもどり値，ネストおよび再帰呼出しはなく，出力は標準関数でのサポートとした． LL(1) 文法適用可能な文法とするため，条件は小括弧で囲むという制約を設けた．まほろば 0 のソースプログラムで記述できる文字は 2 バイトの漢字符号系だけとし, 行と空白の規則はまほろばと同じとした．名前や構文は日本語構文であり，名前にも漢字や仮名が使用できる．

本節では, まほろば 0 の文法を, コンパイラの字句解析で対象とするものと，構文解析で対象とするものとに分けて，バッカス記法で定義する．

B.2.1　字句解析レベルの文法の定義……文字，語，空白と行の規則

字句解析での関心は，プログラム中で使用できる文字と語の構成である．まほろば 0 言語のソースプログラムで記述できる文字は，2 バイトの漢字コードで，まほろば 0 文字集合という．ここで，「…」は連続する自明な文字を省略していることを意味する．

＜まほろば 0 文字集合＞→＜空白＞｜＜語に使う文字＞｜＜記号＞

＜空白＞→△

空白文字を特に表現するときは△を使用する．

＜語に使う文字＞→＜数字＞｜＜英字＞｜＜ハイフン＞｜＜仮名＞
｜＜長音記号＞｜＜漢字＞

＜記号＞→＜ピリオド＞｜＜コンマ＞｜＜コロン＞｜＜演算記号＞
｜＜かぎ括弧＞｜＜小括弧＞｜＜大括弧＞

＜数字＞→０｜１｜…｜９

＜英字＞→ａ｜ｂ｜…｜ｚ｜Ａ｜Ｂ｜…｜Ｚ

＜ハイフン＞→ー

ハイフンは，英語の単語を連結するときに用いる．

＜仮名＞→あ｜い｜…｜ん｜ア｜イ｜…｜ン

仮名には，“ ゃ ”，“ ッ ”，“ ば ”“ ポ ” などの小文字，濁音，半濁音を含む．

＜長音記号＞→ー

長音記号は，“ ニューヨーク ”などで使う仮名の長音である．マイナスとは字形は似ているが，異なる文字である．

＜漢字＞→＜JIS 漢字コード表の漢字＞

＜ピリオド＞→．

＜コンマ＞→，

＜コロン＞→：

コロンは，「番号：整数．」のように名前の定義に使う．

＜演算記号＞→＋｜－｜×｜÷｜＝｜≠｜＞｜＜｜≧｜≦｜←

＜かぎ括弧＞→「｜」

かぎ括弧は，「A←Ｂ＋Ｃ．もどる．」のように，複数の文を囲む．

＜小括弧＞→（｜）
　小括弧は，演算の優先順序を変更したり，関数の引数を囲む.
＜大括弧＞→［｜］
　大括弧は，関数の各部の先頭や最後を示す［関数］［名前］［処理］［終了］に用いる.

語には，利用者が定義する名前と定数，および予約語がある．予約語は，利用者が定義する名前としては利用できない.“ のときには ”などの予約語は，構文規則の中に現れる.
例：名前……公約数，キューリー夫人，
　　定数……１２３

＜名前＞→＜先頭文字＞｛＜後続文字＞｝
＜先頭文字＞→＜英字＞｜＜仮名＞｜＜漢字＞
＜後続文字＞→＜先頭文字＞｜＜数字＞｜＜ハイフン＞｜＜長音記号＞
＜定数＞→＜数字＞｛＜数字＞｝

ソースプログラムは，１行以上の行で構成する．行の終わりは，物理的に文字がない場合か,“ 行の終わりを示す特別な文字 ” で認識する.
空白の規則は日本語では英語と異なる．日本語には,「前の行の最終文字が,次の行の先頭文字に連続する」という性質がある．たとえば，前の行が「佐藤さんは東△」で，次の行が「△京駅に着いた.」の場合,「佐藤さんは東京駅に着いた.」というように，間の空白の存在は意識されない．まほろば 0 の字句解析処理でも, 行の左端部分の空白文字と行の右端部分の空白文字, および“ 行の最後を示す特別な文字 ” は，その存在を無視する.
英語では，語と語を区切るために空白を用いる．日本語でも，行の途中に現れる空白は，分かち書きという文法が適用され，語と語，あるいは文節と文節を区切るために用いられる．まほろば 0 でも，行の途中の空白に限り，語と語の区切り文字として処理する.

B.2.2　構文解析レベルの文法の定義

まほろば０の構文のバッカス記法では，関数名，変数名，定数は字句解析レベルで区別がついているので終端記号として扱い，メタ記号の＜＞では囲まない．また，実際のコーディングで使う［　］は，メタ記号の［　］と区別するために，【　】で表現する．

＜プログラム＞→［＜名前部＞］＜関数＞｛＜関数＞｝
　先頭の＜名前部＞には，複数の関数で共用する変数を記述する．
＜関数＞→＜関数先頭部＞［＜名前部＞］＜処理部＞
　関数は三つの部で構成される．名前部は省略できる．
＜関数先頭部＞→【関数】関数名（[変数名｛，変数名}]）.
　関数先頭部には，関数名と引数を記述する．
＜名前部＞→【名前】{変数名｛，変数名}.}
　名前部では，変数を定義する．まほろば０のデータ型は整数だけなので，
　データ型の指定はしない．
＜処理部＞→【処理】{＜文＞}【終了】
　処理部では，文を０個以上記述する．
＜文＞→＜重文＞｜＜空文＞｜＜呼出し文＞｜＜復帰文＞｜＜代入文＞
　　｜＜条件文＞｜＜繰返し文＞｜＜脱出文＞
　ファイルの入出力は，標準関数で行う．
＜重文＞→「＜文＞｛＜文＞}」
＜空文＞→何もしない．
＜呼出し文＞→関数名（[変数名｛，変数名}]）.
　関数名の直後には，必ず左小括弧“（”がくるので，変数名と区別できる．
　小括弧の中には，引数を書く．
＜復帰文＞→もどる．
＜代入文＞→　変数名←＜式＞.
　例：合計点←国語＋数学＋英語.
＜条件文＞→＜条件＞のときは，＜文＞｛それ以外は，＜文＞}
　例：点数≧６０のときは，評価←合格．それ以外のときは，評価←不合格．

＜繰返し文＞→次を繰り返す．＜重文＞

例：ａ←１．次を繰り返す．「ａ＞１０のときは，ぬける．ａ←ａ＋１．」

＜脱出文＞→ぬける．｜繰返しの先頭にもどる．

脱出文は，繰返し文の中の重文の中だけに記述できる．

＜条件＞→（＜式＞＜比較演算子＞＜式＞）

＜比較演算子＞→＝｜≠｜＞｜≧｜＜｜≦

＜式＞→［＜加減演算子＞］＜項＞｛＜加減演算子＞＜項＞｝

式の中を項と因子の階層に分けたのは，×と÷は＋や－より演算の優先順序が高いことを，構文の定義だけで示すためである．

例：－ａ＋ｂ，　項＋項，　項－項

＜項＞→＜因子＞｛＜乗除演算子＞＜因子＞｝

例：因子×因子，　因子÷因子

＜因子＞→変数名｜定数｜（式）

式の最下位の要素は，変数名か定数である．括弧で優先順序を変更できる．

＜加減演算子＞→＋｜－

＜乗除演算子＞→×｜÷

B.3　目的プログラムの構成と中間語

コンパイラを作成するときに，入力となるソースプログラムの文法決定の次に行うことは，出力仕様を決めることである．すなわち，目的プログラムの構成を決め，それぞれの文に対してどのような目的コードを生成するかを決定する．商用のコンパイラの多くは実際の計算機の機械語を生成するが，まほろば０コンパイラでは単純な構成のまほろば０仮想計算機の機械語を生成する．

目的プログラムの構成が決まると，ソースプログラムの文ごとに生成する中間語を決め，その中間語ごとに目的コードを定める．商用コンパイラの多くは数個以上の中間語をもち，段階的に機械語に近づけていくが，まほろば0コンパイラでは，中間語は１種類しかない．

本節では，仮想計算機の機械語と目的プログラムの構成について説明し，次に中間語と目的コードについて説明する．

B.3.1　まほろば 0 仮想計算機の構成と機能

まほろば 0 仮想計算機は，レジスタもなく，スタックもない計算機で，メモリは命令部とデータ部に分かれる(**図 B.1**)．それぞれ 1 から始まる番地をもち，番地の単位は 1 語で，1 語は 4 バイトである．

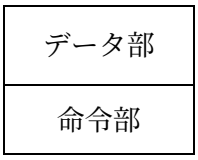


図 B.1　メモリの構成

データ部の構成は，**表 B.1** に示すようにプログラム共通領域とそれぞれの関数の領域とに分かれる．

表 B.1　まほろば 0 仮想計算機のデータ部の構成

種類		内容
プログラム共通領域	固定部	関数のもどり番地領域（1 語） 引数番地リスト（20 語） 演算用作業領域（40 語）
	可変部	共通宣言部の変数領域
それぞれの関数領域	固定部	再帰呼出しフラグ（1 語） もどり番地（1 語）
	可変部	引数番地リスト 名前部の変数領域 関数中で使われた定数領域

命令の構成は，**図 B.2** のように先頭に命令コードがあり，0〜3 個のオペランドが続く．それぞれの長さは，1 語である．オペランドにデータ部の番地が

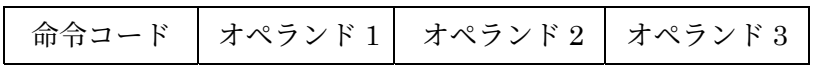


図 B.2　機械語命令の構成

入っている場合，直接指定と間接指定の二通りの番地指定方法がある．直接指定のときは，オペランドに入っている番地が指す領域が操作対象となる．間接指定のときは，オペランドには負の値が入っており，それを絶対値にした値（番地）が，操作対象の領域の番地が入っている領域を指す．前者は，名前部で定義した変数などの参照時に用い，後者は関数の引数で指定された変数の参照時に用いる．これらを，**図 B.3** に図解する．

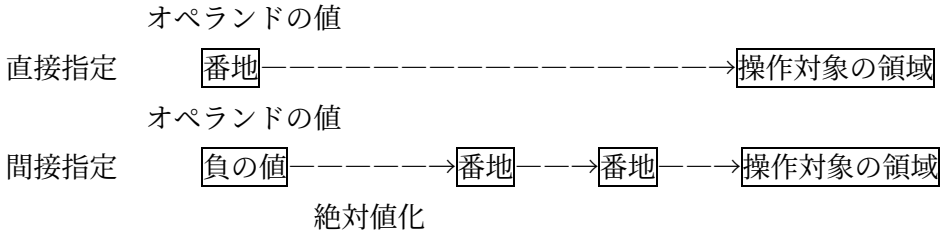


図 B.3　番地参照方式

命令は**表 B.2** に示すように 19 種類ある．（）内は，オペランド数である．

表 B.2　まほろば 0 仮想計算機の機械語命令

命令 (オペランド数)	機能
代入(2)	オペランド 1←オペランド 2．それぞれに操作対象の領域の番地が入る．
定数代入(2)	オペランド 1←（オペランド 2 に入っている整数値）．オペランド 1 には操作対象の領域の番地が入る．
符号反転(1)	オペランド 1←（－（オペランド 1））．オペランド 1 には操作対象領域の番地が入る．
定数加算(2)	オペランド 1←（オペランド 1＋オペランド 2 に入っている整数値）． オペランド 1 は操作対象の領域の番地が入る．
演算(3)	オペランド 1←（オペランド 2　演算子　オペランド 3）．加算，減算，乗算，除算の４種類あり，それぞれの演算子は＋，－，×，÷である．各オペランドには操作対象の領域の番地が入る．
比較(3)	（オペランド 1　比較演算子　オペランド 2）が成立すると，オペランド 3 に入っている飛び先番地に分岐する．等号比較，不等号比較，大記号比較，大等号比較，小記号比較，小等号比較の６種類あり，それぞれの比較演算子は，＝，≠，＞，≧，＜，≦である．
直接分岐(1)	オペランド 1 に入っている飛び先番地に分岐する．
終了（0）	プログラムの実行を終了する．
関数先頭（2）	再帰呼出しフラグ（オペランド 1 に入っている番地の領域）に 1 が入っていれば，“関数が再帰呼出しされた”とのエラーメッセージを出して，プログラムの実行を終了する．1 が入っていなければ，再帰呼出しフラグに 1 を設定する．プログラム共通領域にあるもどり番地と引数番地群を，（オペランド 1 に入っている番地＋1）が指す領域に転記する．オペランド 2 には，引数の個数が入っている．
関数呼出し(1)	関数の引数の番地は，この命令の直前にある命令群で，プログラム共通領域の引数番地リストに入れる．この命令では，直後の命令の番地をプログラム共通領域のもどり番地に設定し，オペランド 1 に入っている呼び出す関数の先頭番地に分岐する．
復帰　(1)	再帰呼出しフラグ（オペランド 1 に入っている番地の領域）に 0 を設定する．もどり番地が，（オペランド 1 に入っている番地＋1）が指す領域に待避されている．そのもどり番地に分岐する．

関数先頭命令，関数呼出し命令，および復帰命令の動作を，**図 B.4** に図解する．

① 関数呼出し命令の直前の命令群は，プログラム共通領域の引数番地に引数の番地を設定する．

② 関数呼出し命令は，プログラム共通領域のもどり番地に自分の直後の番地を設定し，呼び出す関数に分岐する．

③ 呼び出された関数の中から別の関数を呼び出したときに，再びプログラム共通領域が使われるので，この領域は待避しておく必要がある．関数先頭命令で，プログラム共通領域の値を，自分自身の管理する領域（オペランド 1）に移し，後者をその関数の中で参照する．

④ まほろば 0 では関数の再帰呼出しは禁止しているので，そのチェックを関数ごとに用意されている再帰呼出しフラグを用いて行う．このフラグの初期値は 0 であり，関数先頭命令で 1 を設定し，復帰命令で 0 にもどす．関数先頭命令の実行時点で 1 が入っていれば，この関数はまだ実行中であることを意味するので，再帰呼出しとなり，エラー終了する．

⑤ 復帰命令で，自分自身の関数が管理する領域（オペランド 1）からもどり番地を求めて，それに分岐する．

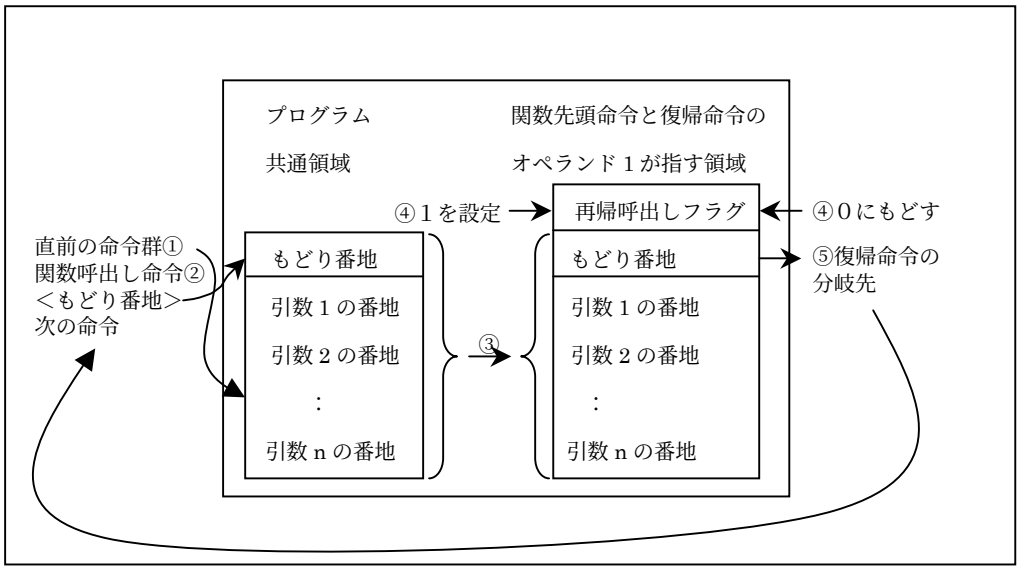


図 B.4　仮想計算機上での関数の呼出しと復帰の動作

B.3.2　中間語とコード生成

まほろば0コンパイラの中間語は17種類あり，命令コードとオペランドで構成される．オペランドは0または1個である．中間語と対応する目的コードを**表B.3**に示す．これらの中間語は，構文解析時に文の種別に対応し生成される．**表B.4**にこの対応を示す．ここで，“ #x ”は変数xの変数名表の登録番号を意味する．ソース上の文と中間語・目的コードとの関連については，**B.5**で説明する．

表 B.3　中間語と対応する目的コード

中間語の命令（オペランド）	目的コード……機械語命令（オペランド）
変数（変数表の番号）	なし．スタック表に変数の番地スタックする．
定数（定数表の番号）	なし．スタック表に定数の番地をスタックする．
文開始（）	なし．スタック表などの文単位の情報を初期化する．
代入（）	代入（スタック表の2番目の番地，スタック表先頭の番地）ただし，第2オペランド（送り側）が演算中間結果の作業領域の場合には，直前の演算命令の第1オペランド（結果格納領域）を第2オペランド（受取り側）で置き換える．
符号反転（）	符号反転（スタック表先頭の番地）
加算（），減算（）， 乗算（），除算（）	演算対応の命令（演算中間結果の作業領域番地，スタック表の2番目の番地，スタック表先頭の番地）
比較（比較演算子種別）	比較命令（スタック表の2番目の番地，スタック表先頭の番地）…第3オペランドは直後の条件分岐で生成する．
条件分岐（タグ番号）	分岐先番地…比較命令の第3オペランドになる．
無条件分岐（タグ番号）	直接分岐（分岐先番地）
タグ定義（タグ番号）	なし
関数定義（関数名表の番号）	関数先頭（関数対応のデータ部先頭番地，引数個数）
引数リスト作成（引数番号）	代入（引数番号に対応するプログラム共通領域の引数番地領域，スタック表の変数の番地）
関数呼出し（関数の先頭番地）	関数呼出し（関数表の番号）
復帰（）	復帰（関数対応のデータ部先頭番地）

表B.4　ソースの文と中間語との対応

ソース上の文	対応する中間語
代入文 A←B	 変数（#A），変数（#B），代入（ ）
A←B+C	変数（#A），変数（#B），変数（#C），加算（ ），代入（ ）
A←B－C×D	変数（#A），変数（#B），変数（#C），変数（#D），乗算（ ）， 減算（ ），代入（ ）
A←－B	変数（#A），変数（#B），符点反転（ ），代入（ ）
条件文 （A＞B÷C） のときは，文 1.	 変数（#A），変数（#B），変数（#C），除算（ ），比較（＞）， 条件分岐（タグ 1），文 1 の中間語， タグ定義（タグ 1）
（A＝B） のときは，文 1 それ以外は， 文 2.	変数（#A），変数（#B），比較（＝），条件分岐（タグ 1）， 文 1 の中間語，無条件分岐（タグ 2）， タグ定義（タグ 1），文 2 の中間語， タグ定義（タグ 2）
繰返し文 次を繰り返す. 「文 1. （A=B）のときは， 先頭にもどる. 文 2. （A=C）のときは， ぬける. 」	 タグ定義（タグ 1）. 文 1 の中間語， 変数（#A），変数（#B），比較（＝），条件分岐（タグ 2）， 無条件分岐（タグ 1）， タグ定義（タグ 2）．文 2 の中間語， 変数（#A），変数（#C），比較（＝），条件分岐（タグ 3）， 無条件分岐（タグ 4）， タグ定義（タグ 3），無条件分岐（タグ 1）， タグ定義（タグ 4）
関数呼出し文 F(A，B)	 変数（#A），引数リスト作成（1），変数（#B），引数リスト作成（2）， 関数呼出し（F の関数表の登録番号）
関数の先頭 F(ａ，ｂ)	 タグ定義（タグ番号） 関数定義（F の関数表の登録番号）
復帰文 もどる.	 復帰（ ）
関数の最後	終了（ ）　関数中に復帰文がないと後続の関数に制御が 移ってしまうので，それを防止する.

B.4　まほろば 0 言語処理系の構成

まほろば0言語処理系は，コンパイラとインタプリタとで構成される（**図5**）．コンパイラは，まほろば 0 ソースプログラムを入力し，目的プログラムとして

仮想計算機の機械語を外部ファイルではなく，メモリ上に生成する．インタプリタは，メモリ上に存在する目的プログラムを解釈実行する．

まほろば 0 コンパイラは，ワンパス・コンパイラであり，まず“ 構文解析フェーズ ”が呼ばれ，それが中心になって構文を解析し，必要に応じ“ 字句解析フェーズ ”と“ 意味解析（中間語生成）フェーズ ”を呼ぶ．“ 字句解析フェーズ ”と“ 意味解析フェーズ ”は，それぞれ“ 読み込みフェーズ ”と“ コード生成フェーズ ”を呼ぶ．この関係を**図 B.6** に示す．

まほろば 0 言語処理系は，32 個の関数で構成されている．これらの関数の関連を**図 B.7** に示す．それぞれの関数を分類すると，**表 B.5** のようになる．

図 B.5　まほろば 0 言語処理系の構成

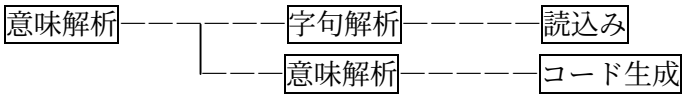
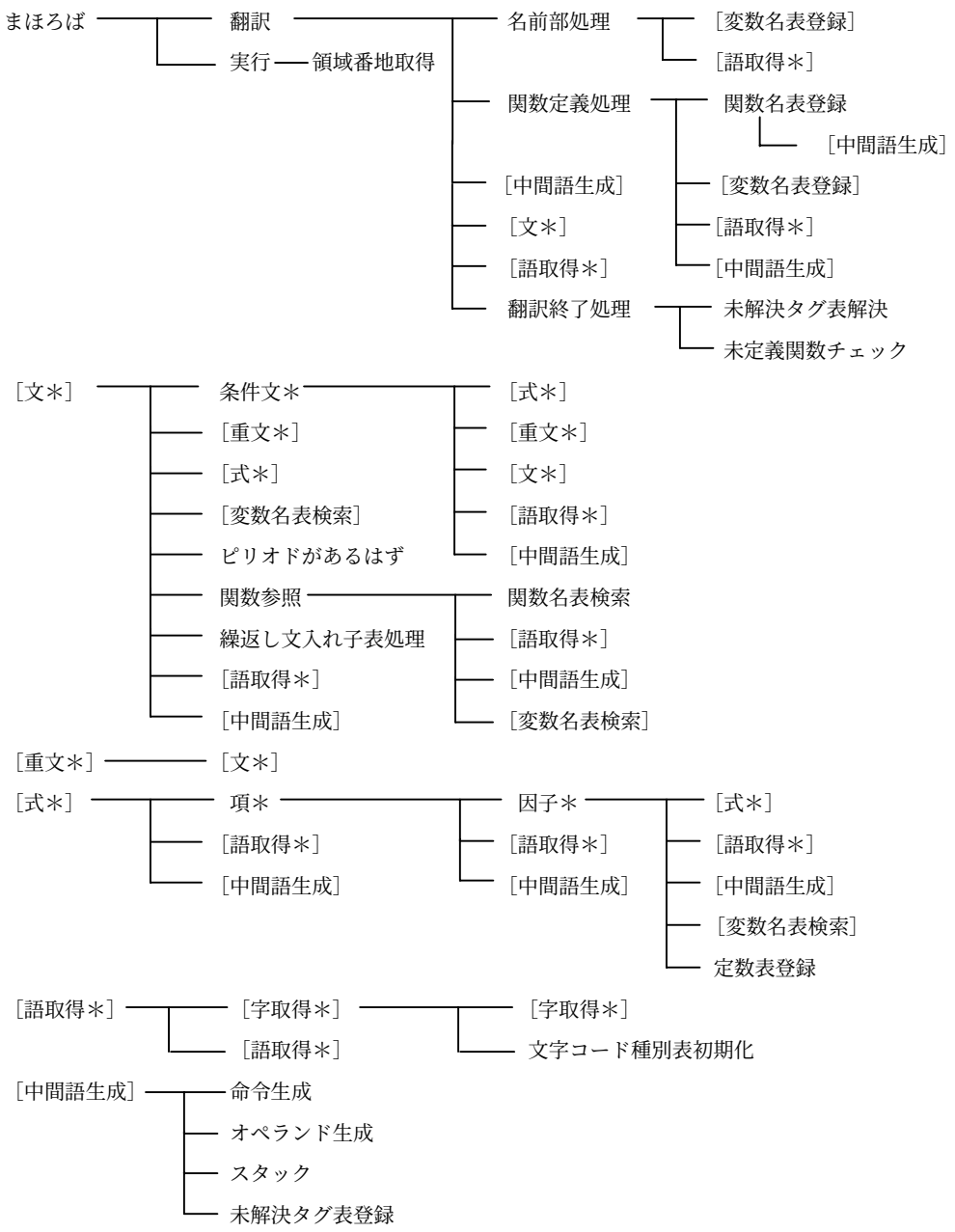


図 B.6　まほろば 0 コンパイラの物理的な構成

表 B.5　まほろば 0 言語処理系のフェーズごとの関数一覧

フェーズ	関数
（共通）	まほろば（メインルーチン），エラー（共通ルーチン）
読込み	字取得，文字コード種別表初期化
字句解析	語取得
構文解析	翻訳，名前部処理，変数名表登録，変数名表検索， 関数定義処理，関数名表登録，関数名表検索，定数表登録， 文，条件文，重文，式，項，因子，関数参照， ピリオドがあるはず，繰返し文入れ子表処理
意味解析	中間語生成，スタック，未定義関数チェック
コード生成	命令生成，オペランド生成，未解決タグ表登録， 未解決タグ表解決，翻訳終了処理
（インタプリタ）	実行，領域番地取得



注：[] で囲んだ関数：　　二つ以上から呼ばれる．

＊のついた関数：　　再帰呼出しされる．

＊＊のついたエラー関数：多くの関数から呼ばれる．呼ぶ側の明示は省略してある．

図 B.7　まほろば 0 コンパイラ関数関連図

B.5　まほろば 0 言語処理系のソースリスト

本節では，まほろば 0 言語処理系の関数のソースリストを**表 B.5** のフェーズの順序で記載する．

B.5.1　共通領域とメイン処理・共通処理関数

プログラム 1　関数間の共通の名前

［名前］　　*{上限値に関係する名前}*

名前けた数上限：定数，３０．

整数けた数上限：定数，３０．

入力行文字数上限：定数，１０００．

ソースファイル名上限：定数，１００．

変数名表上限：定数，１０００．

定数表上限：定数，１０００．

関数名表上限：定数，１００．

引数個数上限：定数，２０．

繰返し文入れ子表上限：定数，３０．

スタック表上限：定数，２０．

タグ表上限：定数，１００００．

未解決タグ表上限：定数，１００００．

目的コード上限：定数，１０００００．

目的コードデータ部固定部分：定数，６１．

目的コードデータ部関数対応固定部分：定数，２．

演算中間結果用作業領域先頭番号：定数，２２．

演算中間結果用作業領域最終番号：定数，６１．

{共通の名前}

成功失敗型：データ型,「失敗，成功」.

空白：定数,“　”.

メッセージ総数：整数，初期値は０．

{文字に関係する名前}

文字：レコード型,「位置：文字位置型．種別：文字種別型．値：文字.」

文字種別型：データ型,「ソースの終わり文字，空白文字，数字，英仮名漢字，ハイフン，
　長音，記号，不当文字」.
文字位置型：データ型,「行，文字位置：整数」.
文字コード種別表［６５５３６］：文字種別型，初期値は不当文字.

｛語に関係する名前｝

語：語型.
語型：データ型,「位置：文字位置型．種別：語種別型．名前［名前けた数上限］：文字.
　詳細：語種別詳細型．整数値：整数.」

｛ 種別だけでは情報不足の場合には，後続項目に値が入る ｝
予約語見出し，演算子，代入記号：語種別詳細型の値.
数値定数整数　　：整数値.
変数名，関数名　　：名前.｝

語種別型：データ型,「予約語部見出し，予約語空文，予約語条件文，予約語それ以外，
　予約語繰返し文，予約語脱出文ぬける，予約語脱出文もどる，予約語復帰文，
　加減演算子，乗除演算子，比較演算子，代入記号，左かぎ括弧，右かぎ括弧，
　左小括弧，右小括弧，ピリオド，コンマ，変数名，関数名，数値定数整数」.
語種別詳細型：データ型,「関数先頭部見出し，名前部見出し，処理部見出し，
　処理部終了見出し，ソースの終わり，加算記号，減算記号，乗算記号，除算記号，
　比較等号，比較不等号，比較大なり，比較大等なり，比較小なり，比較小等なり，
　代入記号」.

｛変数表に関係する名前｝

変数名表［変数名表上限］：レコード型,「名前［名前けた数上限］：文字.
　種別：変数種別型．番地：整数.」
変数種別型：データ型,「引数，変数」.
変数名表登録数：整数，初期値は０.
共通変数最終番号：整数，初期値は０.
現在処理中の関数の変数先頭番号：整数，初期値は０.

｛関数表に関係する名前｝

関数名表［関数名表上限］：レコード型,「名前［名前けた数上限］：文字.
　位置：文字位置型．種別：関数種別型．タグ番号，引数個数，データ部先頭番地：整数.」
関数名表登録数：整数，初期値は０.
関数種別型：データ型,「内部関数定義，関数参照」.

現在の関数名の表番号：整数，初期値は０．

当該関数のデータ部先頭番地：整数，初期値は０．

｛定数表に関係する名前｝

定数表［定数名表上限］：レコード型，「値：整数．番地：整数．」

定数表登録数：整数，初期値は０．

｛繰返し入れ子表に関係する名前｝

繰返し文入れ子表［繰返し文入れ子表上限］：整数．

繰返し文入れ子表番号：整数，初期値は０．

表処理操作種別型：データ型，「登録，削除，検索」．

｛スタック表に関係する名前｝

スタック表［スタック表上限］：整数．

スタッック表登録数：整数，初期値は０．

スタック操作種別型：データ型，「プッシュ，ポップ，クリア」．

｛タグ表に関係する名前｝

タグ表［タグ表上限］：整数．

タグ表登録数：整数，初期値は０．

タグ番号：整数，初期値は０．

未解決タグ表［未解決タグ表上限］：整数．

未解決タグ表登録数：整数，初期値は０．

｛中間語に関係する名前｝

中間語種別型：データ型，「中間語関数定義，中間語引数リスト作成，中間語関数呼出し，
中間語復帰，中間語タグ定義，中間語条件分岐，中間語無条件分岐，中間語変数，
中間語定数，中間語代入，中間語比較，中間語符号反転，中間語加算，中間語減算，
中間語乗算，中間語除算，中間語文開始」．

演算中間結果用作業領域番号：整数，初期値は演算中間結果用作業領域先頭番号．

｛中間語生成に引き渡される中間語は関数“中間語生成”の第１引数（中間語種別）により，次のように第２引数（オペランド）が異なる．

関数定義，関数呼出し：関数表番号

引数リスト作成：引数番号

変数，代入：変数表番号

定数：定数表番号

タグ定義，条件分岐，無条件分岐：タグ番号
比較：比較演算子の種類（語種別詳細型）
その他（復帰，符号反転，加算，減算，乗算，除算，
式開始）：０}
{目的コードに関係する名前}

目的コード：領域共用型，「命令［目的コード上限］：命令種別型，初期値はダミー．
　オペランド［目的コード上限］：整数．」
目的コード生成数：整数，初期値は０．
目的コードデータ部［目的コード上限］：整数，初期値は０．
目的コードデータ部生成数：整数，初期値は目的コードデータ部固定部分．
命令種別型：データ型，「ダミー，関数先頭，関数呼出し，復帰，代入，定数代入，
　符号反転，定数加算，加算，減算，乗算，除算，等号比較，不等号比較，大記号比較，
　大等号比較，小記号比較，小等号比較，直接分岐，終了」．

プログラム２　メイン処理関数と共通処理関数

{次の八つの関数はまほろば０言語処理系で使用する標準関数である．
標準出力，標準入力，ファイルクローズ：もどり値はなし．
ファイルオープン，ファイル入力：もどり値は，成功または失敗．
文字から整数に変換：もどり値は整数．
整数から文字に変換:もどり値はｎ文字,第２引数に文字数を指定．
集団代入：もどり値はなし.}

［関数］まほろば（）．
［処理］翻訳（）．メッセージ総数＝０のときは，実行（）．
［終了］

［関数］エラー（本文［１００］：文字）．
［名前］
出力行：レコード型
　「名なし１：４文字，初期値は“行番号：”．行番号：６文字．
　　名なし２：６文字，初期値は“　文字位置：”．文字位置：４文字．
　　名なし３：文字，初期値は空白．出力本文［１００］：文字．」
出力文字数，文字番号：整数．

［処理］出力行・行番号←整数から文字に変換（語・位置・行，６）.
出力行・文字位置←整数から文字に変換（語・位置・文字位置，４）.
文字番号が１から１００までの間，次を繰り返す.
　「本文［文字番号］が“／”のときは，ぬける.
　　出力本文［文字番号］←本文［文字番号］．」
出力文字数←文字番号＋２０．標準出力（出力行，出力文字数）.
メッセージ総数に１を足す.
［終了］

B.5.2　読込みと字句解析

プログラム 3　ソース入力と文字の取得・類別

［関数］字取得（），再帰可能.
［名前］番号：整数.
呼出し回数：常駐,「最初，２回目以降」，初期値は最初.
｛入力するソース関連の領域｝
入力行［入力行文字数上限］：常駐，文字.
入力文字数，入力行番号，入力文字位置：常駐，整数，初期値は０.
ソースファイル名［ソースファイル名上限］：常駐，文字.
［処理］*｛最初だけ，文字コード種別表の初期化し，ソースファイルをオープンする.｝*
呼出し回数が最初のときは，
　「２回目以降→呼出し回数．文字コード種別表初期化（）.
　標準出力（“ソースファイル名を入力してください．／”）.
　標準入力（ソースファイル名，ソースファイル名上限）.
　ファイルオープン（ソースファイル名）が失敗のときは，
　[1]「エラー（“ソースファイルがオープンできなかった．／”）.
　　文字・位置・行←０．文字・位置・文字位置←０.
　　文字・種別←ソースの終わり．文字・値←空白．もどる.」[1]」
　　｛ソースの入力｝
入力文字位置が０のとき，
　「ファイル入力（ソースファイル名，入力行［］，入力文字数）が失敗のとき，
　[1]「文字・位置・行←入力行番号＋１．文字・位置・文字位置←０.

文字・種別←ソースの終わり．文字・値←空白.

ファイルクローズ（ソースファイル名）．もどる.」$_{1}$

入力行番号に１を足す．入力文字位置←１．

｛行の先頭からの空白と，行の最後からの空白を無視するための処理.｝

番号が１から入力文字数まで，次を繰り返す.

^{1}「入力行［番号］≠空白のときは，ぬける.

入力文字位置に１を足す.

｛ 空白行のときは，自分自身“字取得”を再帰呼出しして，次の行を読む.“字取得”から戻ってきたときは，変数“文字”に値が設定されているので，直ちにもどる.｝

番号＝入力文字数のときは,

^{2}「０→入力文字位置．字取得（）．もどる.」$_{2}$」$_{1}$

番号が入力文字数から入力文字位置まで－１ずつ変化させて，次を繰り返す.

^{1}「入力行［番号］≠空白のときは，ぬける．入力文字数から１を引く.」$_{1}$」

｛文字に値を設定.｝

入力行番号→文字・位置・行．入力文字位置→文字・位置・文字位置.

入力行「入力文字位置」→文字・値.

文字コード種別表［文字から整数に変換（文字・値)］→文字・種別.

｛行の終わりのときは，次回にソース入力するための準備として，入力文字位置に０をセットする.｝

入力文字位置＜入力文字数のときは，入力文字位置に１を足す.

それ以外は，入力文字位置←０.

［終了］

［関数］文字コード種別表初期化（）.

［名前］記号文字［１９］：文字，初期値は“ ，＋－×÷＝≠＞≧＜≦← ［］（）「」”.

［処理］ *｛文字コード種別表の対応する位置に，文字種別型の値を設定.*

英字と仮名漢字については，大量になるので，本論文では省略した.｝

空白→文字コード種別表［文字から整数に変換（“　”)］.

ハイフン→文字コード種別表［文字から整数に変換（“‐”)］.

長音→文字コード種別表［文字から整数に変換（“ー”)］.

マイナス→文字コード種別表［文字から整数に変換（“－”)］.

番号が０から９までの間，次を繰り返す.

「数字→文字コード種別表［文字から整数に変換（“ ０ ”）＋番号］.」

番号が１から１９までの間，次を繰り返す.

「記号→文字コード種別表［文字から整数に変換（“ 記号文字［番号］”）］.」

［終了］

プログラム 4　字句解析（語の取得と語の類別）

［関数］語取得（），再帰可能.

［名前］文字数：整数.

［処理］*｛コンパイラの初期処理で，関数“字取得”と“語取得”を呼んでおく.*

この関数終了時には，次の語の先頭文字は読み込まれているので，この関数が，呼ばれたときには常に，字は１文字先読みにされている．同様に，構文解析処理で語についても常に１語先読みされている.｝

語・位置←文字・位置.

文字・種別の値により分かれる.

「ソースの終わり文字の場合，

[1]「語・種別←予約語部見出し．語・詳細←ソースの終わり．もどる」[1]

空白文字の場合，

[1]「次を繰り返す.

[2]「字取得（）.

文字・種別≠空白文字のときは，[3]「語取得（）．もどる.」[3]」[2]」[1]

数字の場合，

[1]「語・種別←数値定数整数．語・整数値←文字整数変換（文字・値）.

次を繰り返す.

[2]「字取得（）．字・種別≠数字のときは，もどる.

語・整数値←語・整数値×10＋文字整数変換（文字・値）

オーバーフロー時のエラー処理は省略してある.｝　」[2]」[1]

英仮名漢字の場合，

[1]「語・名前［ ］←空白．語・名前［1]←文字・値．文字数←１

次を繰り返す.

[2]「字取得（）.（（字・種別＝英仮名漢字）または（字・種別＝数字）

または（字・種別＝ハイフン）または（字・種別＝長音））のときは，

[3]「文字数に１を足す.
字数≦名前けた数上限のときは，語・名前［文字数]←文字・値.
それ以外は，エラー（“名前が長すぎる．／”）．」[3]
それ以外は,
[3]「 *｛語・名前に入っている名前が，予約語と一致するかチェックする.*
一致するものがあれば，語・種別に対応する値を設定し，もどる.
本論文では，この処理は省略する．｝
字・値＝“（”のときは，語・種別←関数名.
それ以外は，語・種別←変数名.
もどる．」[3]」[2]」[1]
ハイフンの場合,
[1]「エラー（“語の先頭にハイフンがある．／”）．字取得（）．語取得（）.
もどる．」[1]
長音の場合,
[1]「エラー（“語の先頭に長音がある．／”）．字取得（）．語取得（）.
もどる．」[1]
記号の場合,
[1]「 *｛“[”のときは，“関数]”など部の見出しの字句が後続するかチェックする*
続かないときはエラーとし，記号，空白文字，ソースの終わりになるまで，字
をスキップする．“[”でないときは，それぞれの記号に対応する値を，語・
種別と語・詳細に設定する．ここの処理は，本論文では省略する．] 」[1]
不当文字の場合,
[1]「エラー（“不当な文字を検出した．／”）．字取得（）．語取得（）.
もどる．」[1]」
[終了]

B.5.3 構文解析（その 1）

プログラム 5　構文解析（プログラムの構成）

［関数］翻訳（）.
［処理］字取得（）．語取得（）.
語・詳細がソースの終わりのときは,

「エラー（“関数が一つもない．／”）．翻訳終了処理（）．もどる．」

語・種別≠予約語部見出しのときは，

「エラー（“プログラム先頭が部見出しでない．／”）．

語・種別＝予約語部見出しになるまで，次を繰り返す．[1]「語取得（）．」[1]」

語・詳細＝名前部見出しのときは，

「語取得（）．名前部処理（）．変数名表登録数→共通変数最終番号.」

語・詳細＝ソースの終わりになるまで，次を繰り返す．

「語・詳細＝関数先頭部見出しのときは，

[1]「変数名表登録数＋１→現在処理中の関数の変数先頭番号.

当該関数のデータ部先頭番地←目的コードデータ部生成数＋１．

目的コードデータ部生成数に目的コードデータ部関数対応固定部分を足す．

目的コードデータ部生成数＞目的コード上限のときは，

エラー（“データ部が大きすぎる．／”）．

語取得（）．関数定義処理（）．

中間語生成（中間語関数定義，現在の関数名の表番号）.」[1]

それ以外は，

[1]「エラー（“［関数］がない．／”）．語取得（）．

語・種別＝予約語部見出しになるまで，次を繰り返す．[2]「語取得（）.」[2]

繰返しの先頭にもどる.」[1]

語・詳細＝名前部見出しのときは，[1]「語取得（）．名前部処理（）.」[1]

語・詳細＝処理部見出しのときは，

[1]「語取得（）.

語・種別＝予約語部見出しになるまで，次を繰り返す．[2]「文（）.」[2]

語・詳細＝処理部終了見出しのときは，[2]「語取得（）．命令生成（終了）.」[2]

それ以外は，エラー（“［終了］がない．／”）.」[1]

それ以外は，

エラー（“［処理］がない．／”）.」

翻訳終了処理（）.

［終了］

プログラム 6　構文解析（名前部と変数名表）

［関数］名前部処理（）.

［処理］次を繰り返す.

「語・種別の値により分かれる.

[1]「予約語部見出しの場合，もどる.

変数名の場合，変数名表登録（語・名前［］，変数）.

ピリオドの場合，コンマの場合，何もしない.

その他の場合，エラー（“ 変数名でない．／ ”）.」[1]

語取得（）.」

［終了］

［関数］変数名表登録（登録名［名前けた数上限］：文字，変数引数種別：変数種別型）.

［名前］表番号：整数．番地ａ：整数.

［処理］変数名表登録数≧変数名表上限のときは,

「エラー（“ 変数名と引数名が多すぎる．／ ”）．もどる.」

表番号が現在処理中の関数の変数先頭番号から変数名表登録数までの間，次を繰り返す.

「登録名［］＝変数名表・名前［］［表番号］のときは,

[1]「エラー（“ 名前が二重定義．／ ”）．ぬける.」[1]」

変数名表登録数に１を足す.

登録名［］→変数名表・名前［］［変数名表登録数］.

変数引数種別→変数名表・種別［変数名表登録数］.

目的コードデータ部生成数≧目的コード上限のときは,

エラー（“ データ部が大きすぎる．／ ”）.

それ以外は，目的コードデータ部生成数に１を足す.

目的コードデータ部生成数→番地ａ.

変数引数種別が引数のときは，－（番地ａ）→番地ａ.

番地ａ→変数名表・番地［変数名表登録数］.

［終了］

［関数］変数名表検索（検索名［名前けた数上限］：文字），もどり値は整数.

［名前］表番号は，整数.

［処理］表番号が現在処理中の関数の変数先頭番号から変数名表登録数までの間，次を繰り返す.「検索名［］＝変数名表・名前［］［表番号］のときは，もどる（表番号）.」

表番号が，１から共通変数最終番号までの間，次を繰り返す.「検索名［］＝変数名表・

名前［］［表番号］のときは，もどる（表番号）.」

エラー（“ 名前が未定義．／“ ）．もどる（表番号）.

［終了］

プログラム７　構文解析（関数定義と関数表）

［関数］関数定義処理（）.

［名前］語ａ：語型．引数個数：整数，初期値は０.

［処理］語→語ａ.

語・種別＝関数名のときは，語取得（）.

それ以外は，エラー（“ [関数]の直後の名前が関数名ではない．／“ ）.

｛名前の次に左小括弧が続くときは，関数名となり，ここに処理がくる.

よって，左小括弧をチェックせずに，次の語を取得して構わない.｝

次を繰り返す.

「語取得（）.

語・種別の値により分かれる.

[1]「変数名の場合，[2]「引数個数に１を足す．変数名表登録（語・名前［］，引数）.」[2]

コンマの場合，何もしない.

右小括弧の場合，

[2]「語取得（）.

語・種別＝ピリオドのときは，語取得（）.

それ以外は，エラー（“ 関数定義の最後の左小括弧の次にピリオドがない．／”）.

ぬける.」[2]

ピリオドの場合，

[2]「エラー（“ 関数定義のピリオドの前に右小括弧がない．／”）．ぬける.」[2]

予約語部見出しの場合，

[2]「エラー（“ 右小括弧とピリオドがない．／”）．ぬける.」[2]

その他の場合，

[2]「エラー（“ 関数定義の中に不正の語がある．／”）．ぬける.」[2]」[1]」

関数名表登録（語ａ，引数個数）→現在の関数名の表番号.

［終了］

［関数］関数名表登録（関数名：語．引数個数：整数），もどり値は整数．

［名前］表番号は，整数．

一致インジケータ：「一致，不一致」，初期値は不一致．

［処理］表番号が１から関数名表登録数までの間，次を繰り返す．

「関数名・名前［］≧関数名表・名前［］［表番号］のときは，

[1]「関数名表・種別［表番号］＝関数参照のときは，

[2]「一致インジケータ←一致．

関数名表・引数個数［表番号］≠引数個数のときは，

エラー（“関数の引数の個数が不一致．／”）．

ぬける．」[2]」[1]

それ以外は，

[1]「エラー（“関数名が二重定義．／”）．ぬける．」[1]」

一致インジケータが不一致のときは，

「関数名表登録数≧関数名表上限のときは，エラー（“関数名が多すぎる．／”）．

それ以外は，関数名表登録数に１を足す．関数名表登録数→表番号．」

関数名表・名前［］［表番号］←関数名・名前［］．

関数名表・位置［］［表番号］←関数名・位置［］．

関数名表・種別［表番号］←内部関数定義．

一致インジケータが不一致のときは，

「タグ番号に１を足す．関数名表・タグ番号［表番号］←タグ番号．

中間語生成（中間語タグ定義，タグ番号）．」

それ以外は，

中間語生成（中間語タグ定義，関数名表・タグ番号［表番号］）．

関数名表・引数個数［表番号］←引数個数．

関数名表・データ部先頭番地［表番号］←当該関数のデータ部先頭番地．

もどる（表番号）．

［終了］

［関数］関数名表検索（関数名：語．引数個数：整数），もどり値は整数．

［名前］表番号：整数．

［処理］表番号が１から関数名表登録数までの間，次を繰り返す．

「関数名・名前［］＝関数名表・名前［］［表番号］のときは，

[1]「関数名表・引数個数［表番号］≠引数個数のときは,
　　エラー（“ 関数の引数の個数が不一致．／ ”）.
　もどる（表番号）.」[1]」
関数名表登録数＝関数名表上限のときは，エラー（“ 関数名が多すぎる．／ ”）.
それ以外は，関数名表登録数に１を足す.
関数名表・名前［］［関数名表登録数］←関数名・名前［］.
関数名表・位置［］［関数名表登録数］←関数名・位置［］.
関数名表・種別［関数名表登録数］←関数参照.
タグ番号に１を足す.
関数名表・タグ番号［関数名表登録数］←タグ番号.
関数名表・引数個数［関数名表登録数］←引数個数.
もどる（関数名表登録数）.
［終了］

プログラム８　定数表と登録処理

［関数］定数表登録（登録値：整数），もどり値は整数.
［名前］表番号：整数.
［処理］表番号が１から定数表登録数までの間，次を繰り返す.
　「登録値＝定数表・値［表番号］のときは，もどる（表番号）.」
定数表登録数＝定数表上限のときは,
　「エラー（“ 定数が多すぎる．／ ”）．もどる（定数表登録数）.」
定数表登録数に１を足す.
登録値→定数表・値［定数表登録数］.
目的コードデータ部生成数≧目的コード上限のときは,
　エラー（“ データ部が大きすぎる．／ ”）.
それ以外は,
　目的コードデータ部生成数に１を足す.
目的コードデータ部生成数→定数表・番地［定数表登録数］.
登録値→目的コードデータ部［目的コードデータ部生成数］.
もどる（表番号）.
［終了］

B.5.4 構文解析（その 2）

プログラム 9　構文解析（文）

［関数］文（），再帰可能.

［名前］保存タグ番号：整数.

［処理］中間語生成（中間語文開始，０）.

語・種別の値により分かれる.

「　　*{空文}*

予約語空文の場合，語取得（）.

{代入文：変数名が先頭にある文は代入文である.}

変数名の場合，

[1]「中間語生成（中間語変数，変数名表検索（語・名前［］））.

語取得（）.

語・種別≠代入記号のときは，

[2]「エラー（“文の先頭が変数名のときに，後続の語が←でない．／”）.

（語・種別＝ピリオド）または（語・種別＝左かぎ括弧）または

（語・種別＝右かぎ括弧）または（語・種別＝予約語部見出し）になるまで，

次を繰り返す．[3]「「語取得（）.」[3]」[2]

それ以外は，

[2]「語取得（）．式（）.

中間語生成（中間語代入，０）.

ピリオドがあるはず（）.」[2]」[1]

{条件文：左小括弧が先頭にある文は条件文*である.}*

左小括弧の場合，[1]「語取得（）．条件文（）.」[1]

{繰返し文}

予約語繰返し文の場合，

[1]「タグ番号＋２＞タグ表上限のときは，エラー（“条件文や繰返し文が多すぎる.／”）.

それ以外は，タグ番号に２を足す.

タグ番号－１→保存タグ番号.

繰返し文入れ子表処理（登録，保存タグ番号）.

中間語生成（中間語タグ定義，保存タグ番号）.

語取得（）．ピリオドがあるはず（）.

語・種別＝左かぎ括弧のときは，[2]「語取得（）．重文（）.」[2]

それ以外は，エラー（“「次を繰り返す.」の後に，「がない．／”）.

中間語生成（中間語無条件分岐，保存タグ番号）.

中間語生成（中間語タグ定義，保存タグ番号＋１）.

繰返し文入れ子表処理（削除，保存タグ番号）.」[1]

{脱出文}

予約語脱出文ぬけるの場合，

[1]「繰返し文入れ子表処理（検索，保存タグ番号）.

中間語生成（中間語無条件分岐，保存タグ番号＋１）.

語取得（）．ピリオドがあるはず（）.」[1]

予約語脱出文もどるの場合，

[1]「繰返し文入れ子表処理（検索，保存タグ番号）.

中間語生成（中間語無条件分岐，保存タグ番号）.

語取得（）．ピリオドがあるはず（）．」[1]

{呼出し文：関数名が先頭にある文は呼出し文である.}

関数名の場合，[1]「関数参照（）．ピリオドがあるはず（）.」[1]

{重文}

左かぎ括弧の場合，[1]「重文（）.」[1]

{復帰文}

予約語復帰文の場合，

[1]「中間語生成（中間語復帰，現在の関数名の表番号）.

語取得（）．ピリオドがあるはず（）.」[1]

{その他，エラー}

右かぎ括弧の場合，コンマの場合，ピリオドの場合，

[1]「エラー（“ 文の処理中に，誤りの文字を発見した．／”）．語取得（）.」[1]

その他の場合，

[1]「エラー（“ 文の処理中に，誤りの文字を発見した．／”）．語取得（）.」[1]」

［終了］

［関数］重文（），再帰可能.

［処理］次を繰り返す.

「文（）.

語・種別＝右かぎ括弧のときは，

¹「語取得（）．ぬける．」₁

語・種別＝予約語部見出しのときは，

¹「エラー（“重文の処理中に，誤りの文字を発見した．／”）．ぬける．」₁」

［終了］

［関数］ピリオドがあるはず（）．

［処理］語・種別≠ピリオドのときは，エラー（“ピリオドがない．／”）．

（語・種別＝ピリオド）または（語・種別＝左かぎ括弧）または（語・種別＝右かぎ括弧）

または（語・種別＝予約語部見出し）になるまで，次を繰り返す．

「語取得（）．」

語・種別＝ピリオドのときは，語取得（）．

［終了］

プログラム 10　構文解析（条件文）

［関数］条件文（），再帰可能．

［名前］演算子：語種別詳細型．保存タグ番号：整数．

［処理］語取得（）．式（）．

語・種別≠比較演算子のときは，

「エラー（“比較演算子がない．／”）．

（語・種別＝ピリオド）または（語・種別＝左かぎ括弧）または（語・種別＝右かぎ括弧）

または（語・種別＝予約語部見出し）になるまで，次を繰り返す．

¹「語取得（）．」₁もどる．」

演算子←語・詳細．語取得（）．

式（）．

中間語生成（中間語比較，演算子）．

タグ番号≧タグ表上限のときは，エラー（“条件文や繰返し文が多すぎる．／”）．

それ以外は，タグ番号に１を足す．

タグ番号→保存タグ番号．

中間語生成（中間語条件分岐，保存タグ番号）．

語・種別＝右小括弧のときは，語取得（）．

それ以外は，エラー（“右小括弧がない．／”）．

語・種別＝予約語条件文のときは，語取得（）.
それ以外は，エラー（“「のときは」がない．／”）.
語・種別＝コンマのときは，語取得（）.
それ以外は，エラー（“ コンマがない．／”）.
語・種別＝左かぎ括弧のときは，「語取得（）．重文（）.」
それ以外は，文（）.

{「それ以外は」以降の処理}

語・種別＝予約語それ以外のときは，
　「語取得（）.
　　語・種別＝コンマのときは，語取得（）.
　　それ以外は，エラー（“ コンマがない．／”）
　　タグ番号≧タグ表上限のときは，エラー（“ 条件文や繰返し文が多すぎる．／”）.
　　それ以外は，タグ番号に１を足す.
　　中間語生成（中間語無条件分岐，タグ番号）.
　　中間語生成（中間語タグ定義，保存タグ番号）.
　　タグ番号→保存タグ番号.
　　語種別＝左かぎ括弧のときは，[1]「語取得（）．重文（）.」[1]
　　それ以外は，文（）.」
それ以外は，
　中間語生成（中間語タグ定義，保存タグ番号）.
［終了］

プログラム 11　構文解析（式）

［関数］式（），再帰可能.
［名前］語 a：語種別型.
［処理］語 a ←語．語・種別が加減演算子のときは，
　「語取得（）．項（）.
　　語 a・詳細が減算記号のときは，中間語生成（中間語符号反転，０）.」
それ以外は，
　項（）.
語 a←語.
語 a・種別が加減演算子の間，次を繰り返す.

「語取得（）．項（）．
　語ａ・詳細が減算記号のときは，中間語生成（中間語減算，０）．
　それ以外は，中間語生成（中間語加算，０）．
　語 a←語.」
［終了］

［関数］項（），再帰可能.

［名前］語ａ：語種別型.
［処理］因子（）．語 a←語.
語ａ・種別が乗除演算子の間，次を繰り返す.
「語取得（）．因子（）.
　語ａ・詳細が乗算記号のときは，中間語生成（中間語乗算，０）.
　それ以外は，中間語生成（中間語除算，０）.
　語 a←語.」
［終了］

［関数］因子（），再帰可能.

［処理］語・種別の値により分かれる.
「変数名の場合，
　[1]「中間語生成（中間語変数，変数名表検索（語・名前［］））．語取得（）.」[1]
　数値定数整数の場合，
　[1]「中間語生成（中間語定数，定数表登録（語・整数値））．語取得（）.」[1]
　左小括弧の場合，
　[1]「語取得（）．式（）.
　　語・種別＝右小括弧のときは，語取得（）.
　　それ以外は，エラー（“対応する右括弧がない．／”）.」[1]」
［終了］

プログラム 12　構文解析（関数参照）

［関数］関数参照（）.

［名前］語ａ：語型．引数番号：整数，初期値は０.
［処理］語→語ａ．語取得（）.

｛文の先頭で名前の次に左小括弧が続くときは，それは関数名であり，ここに処理がくる．左小括弧をチェックせずに，次の語を取得して構わない.｝

次を繰り返す.

「語取得（）.

語・種別の値により分かれる.

[1]「変数名の場合,

[2]「引数番号に１を足す.

中間語生成（中間語変数，変数名表検索（語・名前［］））.

中間語生成（中間語引数リスト作成，引数番号）.」[2]

コンマの場合，何もしない.

右小括弧の場合,

[2]「語取得（）.

語・種別＝ピリオドのときは，語取得（）」[2].

それ以外は，エラー（“関数参照の最後にある左小括弧の次にピリオドがない.／”）.

ぬける.」[2]

ピリオドの場合，[2]「エラー（“右小括弧がない．／”）．語ぬける.」[2]

予約語部見出しの場合，[2]「エラー（“右小括弧とピリオドがない．／”）．ぬける.」[2]

その他の場合，[2]「エラー（“関数参照の中に不正の語がある．／”）．ぬける.」[2]」[1]」

中間語生成（中間語関数呼出し，関数名表検索（語ａ，引数番号））.

［終了］

プログラム 13　構文解析（繰返し文入れ子表）

［関数］繰返し文入れ子表処理（操作：表処理操作種別型，保存タグ番号：整数）.

［処理］操作の値により分かれる,

「登録の場合,

[1]「繰返し文入れ子表上限＞繰返し文入れ子表番号のときは,

繰返し文入れ子表番号に１を足す.

それ以外は,

エラー（“繰返し文のネストが深すぎる．／”）.

繰返し文入れ子表［繰返し文入れ子表番号］←保存タグ番号.」[1]

削除の場合,

[1]「繰返し文入れ子表番号＞０のときは,

繰返し文入れ子表番号から１を引く.

それ以外は,

エラー（“繰返し文ネスト処理不良（システムエラー）．／”）.」[1]

検索の場合,

[1]「繰返し文入れ子表番号＞０のときは,

保存タグ番号←繰返し文入れ子表［繰返し文入れ子表番号］.

それ以外は,

エラー（“繰返し文ネスト表の状態不良（システムエラー）．／”）.」[1]」

［終了］

B.5.5　意味解析

プログラム 14　中間語処理と目的コード生成

［関数］中間語生成（中間語命令：中間語種別型，中間語オペランド：整数）.

［名前］番地ａ，番地ｂ：整数.

［処理］中間語命令の値により分かれる.

「中間語関数定義の場合,

[1]「命令生成（関数先頭）.

オペランド生成（関数名表・データ部先頭番地［中間語オペランド］）.

オペランド生成（関数名表・引数個数［中間語オペランド］）.」[1]

中間語引数リスト作成の場合,

[1]「命令生成（代入）.

オペランド生成（中間語オペランド＋１）.

スタック（ポップ，番地ａ）.

オペランド生成（番地ａ）.」[1]

中間語関数呼出しの場合,

[1]「命令生成（関数呼出し）.

オペランド生成（関数名表・タグ番号［中間語オペランド］）.

未解決タグ表登録（）.」[1]

中間語復帰の場合,

[1]「命令生成（復帰）.

オペランド生成（関数名表・データ部先頭番地［中間語オペランド］）.」$_{1}$

中間語タグ定義の場合,

目的コード生成数→タグ表［中間語オペランド］.

中間語条件分岐の場合,

^{1}「オペランド生成（中間語オペランド）.

未解決タグ表登録（）.」$_{1}$

中間語無条件分岐の場合,

^{1}「命令生成（直接分岐）.

オペランド生成（中間語オペランド）.

未解決タグ表登録（）.」$_{1}$

中間語変数の場合,

スタック（プッシュ（変数名表・番地［中間語オペランド］））.

中間語定数の場合,

スタック（プッシュ（定数表・番地［中間語オペランド］））.

中間語代入の場合,　　*｛番地ａ←番地ｂ｝*

^{1}「スタック（ポップ，番地ｂ）.

スタック（ポップ，番地ａ）.

（番地ｂ≧演算中間結果用作業領域先頭番号）および（番地ｂ≦演算中間結果用作業領域最終番号）および（番地ｂ＝目的コード・オペランド［目的コード生成数－３］）のときは,

番地ａ→目的コード・オペランド［目的コード生成数－３］.

｛送り側（番地ｂ）が作業領域で，それが直前の演算の第１オペランド（間結果領域）のときは，中間結果領域は代入の受け取り側（番地ａ）は置き換え可能である.｝

それ以外は,

^{2}「命令生成（代入）.

オペランド生成（番地ａ）.

オペランド生成（番地ｂ）.」$_{2}$」$_{1}$

中間語比較の場合,　　*｛番地ａ　☆　番地ｂ｝*

^{1}「命令生成（等号比較＋（中間語オペランド－比較等号））.

スタック（ポップ，番地ｂ）.

スタック（ポップ，番地ａ）.

オペランド生成（番地ａ）.

オペランド生成（番地ｂ）.」1

中間語符号反転の場合,

1「命令生成（符号反転）.

スタック（ポップ，番地ａ）.

オペランド生成（番地ａ）.」1

中間語加算の場合，中間語減算の場合，中間語乗算の場合，中間語除算の場合,

[受取り側←　番地ａ　☆　番地ｂ]

1「スタック（ポップ，番地ｂ）.

スタック（ポップ，番地ａ）.

命令生成（加算＋（中間語オペランド－中間語加算））.

演算中間結果用作業領域番号≧演算中間結果用作業領域最終番号のときは,

エラー（“式が複雑すぎる．／”）.

それ以外は,

演算中間結果用作業領域番号に１を足す.

スタック（プッシュ，演算中間結果用作作業領域番号）.

オペランド生成（演算中間結果用作作業領域番号）.

オペランド生成（番地ａ）．オペランド生成（番地ｂ）.」1

中間語文開始の場合,

1「スタック（クリア，０）.

演算中間結果用作業領域番号←演算中間結果用作業領域先頭番号.」1」

［終了］

プログラム 15　式の中の変数・定数のスタックと取り出し

［関数］スタック（操作：スタック操作種別型，オペランド番号：整数）.

［処理］操作の値により分かれる.

「プッシュの場合,

1「スタック表登録数≧スタック表上限のときは,

エラー（“式の中の項目が多すぎる．／”）.

それ以外は,

2「スタック表登録数に１を足す.

オペランド番号→スタック表［スタック表登録数］.」2」1

ポップの場合,

[1]「スタック表登録数≦０のときは,

エラー（“ スタック表が空（システムエラー）．／ ”）.

それ以外は,

[2]「スタック表［スタック表登録数］→オペランド番号.

スタック表登録数から１を引く.」[2]」[1]

クリアの場合,

０→スタック表登録数.」

［終了］

プログラム 16　未定義関数のチェック

［関数］未定義関数チェック（）.

［名前］表番号：整数.

［処理］表番号が１から関数名表登録数までの間，次を繰り返す.

「関数名表・種別［表番号］＝関数参照のときは,

[1]「 *{標準関数の場合の処理は､本論文では省略している.}*

語・位置←関数名表・位置［表番号］.

エラー（“ この関数の定義がない．／ ”）.」[1]」

［終了］

B.5.6　コード生成

プログラム 17　コード生成

［関数］命令生成（命令部：命令種別型）.

［処理］目的コード生成数≧目的コード上限のときは,

エラー（“ 文が多いので，目的コードが上限を超えた．／ ”）.

それ以外は,

目的コード生成数に１を足す.

命令部→目的コード・命令［目的コード生成数］.

［終了］

［関数］オペランド生成（オペランド：整数）.

［処理］目的コード生成数≧目的コード上限のときは,

　エラー（“ 文が多いので，目的コードが上限を超えた．／ ”）.

それ以外は,

　目的コード生成数に１を足す.

オペランド→目的コード・オペランド［目的コード生成数］.

［終了］

プログラム 18　未解決の飛び先（タグ）の番地の解決

［関数］未解決タグ表登録（）.

［処理］未解決タグ表登録数≧未解決タグ表上限のときは,

　エラー（“ プログラム中の分岐が多すぎる．／ ”）.

それ以外は,

　未解決タグ表登録数に１を足す.

目的コード生成数→未解決タグ表［未解決タグ表登録数］.

［終了］

［関数］未解決タグ表解決（）.

［名前］表番号，目的コード番号，タグ番号：整数.

［処理］表番号が１から未解決タグ表登録数までの間，次を繰り返す.

「未解決タグ表［表番号］→目的コード番号.

　目的コード・オペランド［目的コード番号］→タグ番号.

　タグ表［タグ番号］→目的コード・オペランド［目的コード番号］.」

［終了］

プログラム 19　翻訳の終了処理

［関数］翻訳終了処理（）.

［名前］表番号：整数.

［処理］未解決タグ表解決（）．未定義関数チェック（）.

［終了］

B.5.7　インタプリタ

プログラム 20　まほろば０のインタプリタ

［関数］実行（）.

［名前］番地：整数，初期値は１．オペランド１，　オペランド２，オペランド３：整数.

［処理］番地が０になるまで，次を繰り返す.

「目的コード・命令［番地］の値により分かれる.

[1]「関数先頭の場合,

[2]「オペランド１←目的コード・オペランド［番地＋１］.

目的コードデータ部［オペランド１］＝１のときは,

[3]「標準出力（“関数が再帰呼出しされた．プログラムの実行は中止する．”,

２６）．０→番地.」[3]

それ以外は,

[3]「目的コードデータ部［オペランド１］←１.

｛集団代入関数は，標準関数｝

集団代入（目的コードデータ部［オペランド１＋１］,

目的コードデータ部［１］，目的コード・オペランド［番地＋２］＋１）.

番地に３を足す.」[3]」[2]

関数呼出しの場合,

[2]「番地＋２→目的コードデータ部［１］.

番地←目的コード・オペランド［番地＋１］.」[2]

復帰の場合,

[2]「目的コードデータ部［目的コード・オペランド［番地＋１］］←０.

番地←目的コードデータ部［目的コード・オペランド［番地＋１］＋１］.

｛先頭の関数中からもどるときは，飛び先の領域には０が入っているので，プログラムは終了できる.｝　」[2]

代入の場合,

[2]「オペランド１←領域番地取得（番地＋１）.

オペランド２←領域番地取得（番地＋２）.

目的コードデータ部［オペランド１］←目的コードデータ部［オペランド２］.

番地に３を足す.」[2]

定数代入の場合,

[2]「オペランド１←領域番地取得（番地＋１）.
目的コードデータ部［オペランド１］←目的コード・オペランド［番地＋２］」.
番地に３を足す.」[2]
符号反転の場合,
[2]「オペランド１←領域番地取得（番地＋１）.
目的コードデータ部［オペランド１］←－目的コードデータ部［オペランド１］.
番地に２を足す.」[2]
定数加算の場合,
[2]「オペランド１←領域番地取得（番地＋１）.
目的コードデータ部［オペランド１］に目的コード・オペランド［番地＋２］」
を足す.
番地に３を足す.」[2]
加算の場合,
[2]「オペランド１←領域番地取得（番地＋１）.
オペランド２←領域番地取得（番地＋２）.
オペランド３←領域番地取得（番地＋３）.
目的コードデータ部［オペランド１］←
目的コードデータ部［オペランド２］＋目的コードデータ部［オペランド３］.
番地に４を足す.」[2]
減算の場合,
[2]「オペランド１←領域番地取得（番地＋１）.
オペランド２←領域番地取得（番地＋２）.
オペランド３←領域番地取得（番地＋３）.
目的コードデータ部［オペランド１］←
目的コードデータ部［オペランド２］－目的コードデータ部［オペランド３］.
番地に４を足す.」[2]
乗算の場合,
[2]「オペランド１←領域番地取得（番地＋１）.
オペランド２←領域番地取得（番地＋２）.
オペランド３←領域番地取得（番地＋３）.
目的コードデータ部［オペランド１］←
目的コードデータ部［オペランド２］×目的コードデータ部［オペランド３］.

番地に４を足す.」$_2$

除算の場合,

^{2}「オペランド１←領域番地取得（番地＋１).

オペランド２←領域番地取得（番地＋２).

オペランド３←領域番地取得（番地＋３).

目的コードデータ部［オペランド１］←

目的コードデータ部［オペランド２］÷目的コードデータ部［オペランド３].

番地に４を足す.」$_2$

等号比較の場合,

^{2}「オペランド１←領域番地取得（番地＋１).

オペランド２←領域番地取得（番地＋２).

（目的コードデータ部［オペランド１］＝目的コードデータ部［オペランド２］）

のときは，番地に４を足す.

それ以外は，番地←目的コード・オペランド［番地＋３].」$_2$

不等号比較の場合,

^{2}「オペランド１←領域番地取得（番地＋１).

オペランド２←領域番地取得（番地＋２).

（目的コードデータ部［オペランド１］≠目的コードデータ部［オペランド２］）

のときは，番地に４を足す.

それ以外は，番地←目的コード・オペランド［番地＋３].」$_2$

大記号比較の場合,

^{2}「オペランド１←領域番地取得（番地＋１).

オペランド２←領域番地取得（番地＋２).

（目的コードデータ部［オペランド１］＞目的コードデータ部［オペランド２］）

のときは，番地に４を足す.

それ以外は，番地←目的コード・オペランド［番地＋３].」$_2$

大等号比較の場合,

^{2}「オペランド１←領域番地取得（番地＋１).

オペランド２←領域番地取得（番地＋２).

（目的コードデータ部［オペランド１］≧目的コードデータ部［オペランド２］）

のときは，番地に４を足す.

それ以外は，番地←目的コード・オペランド［番地＋３].」$_2$

小記号比較の場合,

^{2}「オペランド１←領域番地取得（番地＋１）.

オペランド２←領域番地取得（番地＋２）.

（目的コードデータ部［オペランド１］＜目的コードデータ部［オペランド２］）

のときは，番地に４を足す.

それ以外は，番地←目的コード・オペランド［番地＋３］.」$_{2}$

小等号比較の場合,

^{2}「オペランド１←領域番地取得（番地＋１）.

オペランド２←領域番地取得（番地＋２）.

（目的コードデータ部［オペランド１］≦目的コードデータ部［オペランド２］）

のときは，番地に４を足す.

それ以外は，番地←目的コード・オペランド［番地＋３］.」$_{2}$

直接分岐の場合，番地←目的コード・オペランド［番地＋１］.

終了の場合,

^{2}「エラー（“復帰文を実行していないのに，関数の終わりに達した．／”）.

番地←０.」$_{2}$

それ以外の場合,

^{2}「エラー（“実行時システムエラー．目的コードの命令の値がおかしい．／”）.

番地←０.」$_{2}$」$_{1}$」

［終了］

［関数］領域番地取得（オペランド番号：整数），もどり値は整数.

｛オペランドが変数のときは，目的コード・オペランド［オペランド番号］が直接，その領域を指す．オペランドが関数の引数のときは，目的コード・オペランド［オペランド番号］は負になっており，その絶対値が指す領域に，引数の領域の番地が入っている.｝

［処理］目的コード・オペランド［オペランド番号］＞０のときは,

もどる（目的コード・オペランド［オペランド番号］）.

それ以外は,

もどる（目的コードデータ部［－（目的コード・オペランド［オペランド番号］）］）.

［終了］

B.6 まとめ

本付録では，日本語プログラム言語“まほろば”の実用性評価のために作成したまほろば 0 言語の文法と処理系についてソースリストをつけて紹介した．これは，大学でコンパイラを教えるときの教材として手頃な規模であり，その教科書[今城 2000]はすでに出版済みである．

謝辞

本研究を行うにあたりご指導および励ましいただいた奈良先端科学技術大学院大学植村俊亮教授に深く感謝申し上げます．植村先生には 30 年以上前から COBOL 標準化作業でご指導いただき，さらに 50 歳を超えた筆者を植村研究室でお引き受けいただき本研究の主指導教官になっていただきました．

本研究の副指導教官になっていただいた奈良先端科学技術大学院大学渡邉勝正教授，松本健一教授，吉川正俊助教授に感謝申し上げます．

また下記の方々をはじめ多くの方からご指導，激励いただきました．ありがとうございました．

5 年前に奈良先端科学技術大学院大学鳥居宏次教授（現在は学長）から社会人の大学院入学制度があることを教えていただき，東京農工大学吉澤康文教授と高橋延匡教授（現在は拓殖大学）からは博士論文を執筆するようお奨めいただきました．これがきっかけになり本研究にたずさわることができました．

日立製作所ソフトウェア事業部の小平光彦事業部長（現在は日立情報システムズ常務取締役），山本章治事業部長（現在は日立ソフトウェアエンジニアリング常務取締役），柴宮実事業部長，山本義幸販売推進本部長，小塚潔ネットワークソフトウェア本部長には，勤務のかたわら大学院に社会人入学することをお許しいただき，終始励ましていただきました．

東京農工大学西村恕彦名誉教授，法政大学中田育男教授，慶応義塾大学土居範久教授，早稲田大学筧捷彦教授にはプログラム言語の標準化活動において長い間ご指導いただきました．また，あまりにも多いので一部の方以外の名前は省かせていただきますが，国内および海外の標準化委員会の多くの委員との国・企業・大学の枠をこえた共同作業で，多くのことを学びました．この中で，次の方々と一緒に推進した COBOL 日本語機能および国際化の標準化作業の経験が，本論文第 2 章の基になっています．

Ms. Ann Bennett（ISO COBOL 標準化委員会(ISO/IEC JTC1/SC22/WG4) 委員長, IBM），Mr. Don Schricker(米国 COBOL 標準化委員会（INCITS/J4）

委員長，Micro Focus)，植村俊亮教授（奈良先端科学技術大学院大学)，菊池道夫氏（日本 IBM)，黒田幸明氏（NTT)，小林純一氏（沖電気工業，現在は Micro Focus)，五月女健治氏（三菱電機)，佐藤孝夫氏（日本電気)，菅田和男氏（富士通)，高木渉氏（日立製作所)，塚越敏晴氏（日立製作所)，床分眞一氏（日立製作所)，花田良平氏（日立ソフトウェアエンジニアリング)，三宅立記氏（日立ソフトウェアエンジニアリング)，宮本道夫氏（日本ユニバック)，山谷祐二氏（東芝)．

姫路工業大学宮脇富士夫名誉教授，東京大学玉井哲雄教授，慶応義塾大学大岩元教授，神奈川大学教授野口健一郎教授，芝浦工業大学古宮誠一教授，都立航空工業高等専門学校鈴木弘助教授，文部省メディア研究センター研究開発部大澤範高氏，慶応義塾大学大学院中鉢欣秀氏，キヤノンソフトウェア増田和紀氏には，日本語プログラム言語に関する参考資料を提供いただくなど，ご教示・ご協力いただきました．

本研究の基となった日立製作所でのプログラム言語，コンパイラおよびシステム開発支援ツールの開発成果は，システムソリューショングループ生産技術統括本部の大野治本部長，ソフトウェア事業部の言語設計部床分眞一技師，第 2 ネットワークソフト設計部吉野松樹部長，ソフトウェアサービス推進センタ秋山美登センタ長，日立システムアンドサービスの産業・流通システムサービス事業部東京産業・流通本部津田道夫本部長，日立ソフトウェアエンジニアリングの開発事業部九州開発センタ三宅立記センタ長および脇坂隆則主任技師など多くの同僚との共同作業によるもので，いろいろ教えていただきました．

本論文の内容について，日立製作所ソフトウェア事業部の大場みち子主任技師には，博士論文をどう書くかの立場からレビューいただき，日立インフォメーションテクノロジーのサーバ開発本部難波靖徳エンジニアには，まほろば 0 処理系の論理チェックをしていただきました．湘南サービスの松本智子さんにはワープロ作業を手伝っていただきました．

最後に，本研究は，妻の美枝子と娘の智子および澄子の理解と支えによって推進することができました．心から感謝します．

参考文献

あ

[阿部 87]　阿部昇，市丸数馬，渡辺照洋（富士通）：C 言語での日本語サポート方式，情報処理学会第 35 回（昭和 62 年後期）全国大会，7R-5，pp.901-902，1987.

[アルゴリズム辞典 94]　アルゴリズム辞典，pp.587-589，共立出版，1994.

[石井 99]　石井義之，畠山正行（茨城大），加藤木和夫（日立プロセスコンピュータエンジニアリング）：オブジェクト指向日本語分析記述環境 OOJ の設計開発, 情報処理学会研究報告（第 122 回ソフトウェア工学研究会）, 99-SE-122, pp.55-62，1999 年 3 月 18 日.

[石田 83]　石田晴久（東大）：ドキュメント作成の現状と将来，電子通信学会誌（ソフトウェア生産技術の現状と将来特集号），Vol.66，No.4，pp.402-407，1983.

[伊集院 88]　伊集院正（NTT），石畑清（東大）：Ada における日本語化の現状と今後の方向, 情報処理学会プログラミング言語研究会資料（16-11）, pp.61-64, 1988-5-13

[稲田 86]　稲田満，岡本務，渡辺敏，中村雄三（NTT）：プログラム設計用言語 SL の開発と評価, 情報処理学会論文誌, Vol.27，No.6，pp.601-611, 1986.

[猪瀬 88]　猪瀬武久（日電）：C における日本語化の現状と今後の方向，情報処理学会プログラミング言語研究会資料（16-10），pp. 57-60，1988-5-13.

[今城 89]　今城哲二，秋山美登（日立），北尾修二，里本健，脇坂隆則（日立ソフトウェアエンジニアリング），大西俊治（日立）：VOS K 第 4 世代言語 “EAGLE/4GL”，日立評論，Vol.71，No.11，pp.1119-1124，1989.

[今城 90a]　Tetsuji Imajo, Yoshinori Akiyama（日立）and Shuji Kitao（日立ソフトウェアエンジニアリング）：VOS K 4th Generation Language “EAGLE/4GL”，Hitachi Review, Vol.39, No.5, pp.261-266, 1990.

[今城 90b]　今城哲二（日立）：明解 COBOL, 213 ページ, サイエンス社, 1990.

[今城 99a]　今城哲二（日立／奈良先端大）：日本語プログラム言語文献ノート，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.9-16，情報処理学会，1999.

[今城 99b]　今城哲二（日立／奈良先端大）：日本語プログラム言語の設計，第 40 回プログラム・シンポジウム報告集，情報処理学会，pp.7-19，1999-1-12.
[今城 99c]　今城哲二，秋藤俊介，亀頭政義，辻畑好秀（日立）：日本からの ISO と OMG での標準化提案事例とその海外展開，電子情報通信学会第 5 回ソフトウェアグローバル競争力研究会，電子情報通信学会技術研究報告，SGC 99-17〜20 第二分冊 〔ソフトウェアグローバル競争力〕，pp.1-10，1998 年 1 月 18 日.
[今城99d]　今城哲二（日立）：＜標準活動トピックス＞標準化プログラム言語の国際化，情報技術標準 No.44，pp.2-6，情報処理学会情報規格調査会，1999年12月.
[今城 2000]　今城哲二（日立）：第 4 章　言語処理系，入門計算機ソフトウェア（金子敬一（東京農工大），今城哲二（日立），中村英夫（日大）共著），pp.76-130，朝倉書店，2000-4.
[今城 2001a]　今城哲二（日立／奈良先端大），鈴木弘（都立航空工業高専），大野治（日立），植村俊亮（奈良先端大）：日本語プログラム言語 “ まほろば ” の言語仕様，情報処理学会研究報告（ソフトウェア工学研究会），2000-SE-130，pp.143-152，2001-3.
[今城 2001b]　今城哲二（日立／奈良先端大），吉野松樹，大坪稔房，原田晃，大野治（日立），植村俊亮（奈良先端大）：オンライン業務プログラムの環境独立処理方式，情報処理学会研究報告（ソフトウェア工学研究会），2000-SE-135，pp.33-40，2001 年 11 月 21 日.
[岩波辞典 90]　岩波情報科学辞典，p.551 及び p.720，岩波書店，1990.
[岩元 87]　岩元莞二，西谷泰昭，和田孝（日電）：プログラム自動生成システム，NEC 技報，Vol.40，No.1，pp.35-38，1987.
[植木 88]　植木定裕，宮内和人，中田晴美（富士通）：高生産ツール CASET，FUJITSU，Vol.39，No.1，pp.21-28，1988.
[上田 2002]　上田賀一，畠山正行（茨城大），加藤木和夫（日立情報制御システム）：オブジェクト指向計算記述日本語 OCDJ の設計，信学技報（電子情報通信学会技術研究報告［ソフトウェアサイエンス］），Vol.101，No.629，pp.39-46，2002.
[上原邦昭 86]　上原邦昭（阪大），藤井邦和（日本 IBM），豊田順一（阪大）：

自然言語による自動プログラム合成, コンピュータソフトウェア, Vol.3, No.4, pp.359-368，1986.
[上原憲二 88]　上原憲二，黒田清隆，土井日輝，行徳孝彦，鈴木由美子（三菱電機）：プログラム自動生成システム SAGE，情報処理学会ソフトウェア工学研究会資料（58-10），pp.73-79，1988-2-4.
[上原三八 83]　上原三八，佐藤秀樹，毛利友治，小野越夫，高尾哲康（富士通研）：詳細設計支援システム PDAS におけるプログラム生成，情報処理学会第 27 回（昭和 58 年後期）全国大会，3C-3，pp.497-498，1983.
[宇土 88]　宇土正浩，佐貫俊幸（日本 IBM）：日本語 APL，情報処理学会プログラミング言語研究会資料（16-3），pp.17-23，1988-5-13.
[大石 88]　大石東作（電総研）：ソフトウェアの「日本語仕様書」作成支援，情報処理学会ソフトウェア工学研究会資料（59-4），7 ページ，1988-5-18.
[大澤 93]　大澤範高（パーソナルメディア），木村憲雄（東大）：プログラム言語「きなり」，トロン技術研究会，Vol.5，No.2，pp.39-50，1993-3.
[大澤 95]　大澤範高，弓場敏嗣（電通大）：並列離散事象シミュレーション言語「もえぎ」の構想，信学技報（COMP94-92），pp.25-32，1995-3.
[大澤 99]　大澤範高（文部省メディア教育開発センター）：日本語風表記でプログラミングできる言語「もえぎ」とその処理系，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.1-6，情報処理学会，1999.
[大西 87]　大西淳，阿草清滋，大野豊（京大）：要求定義のための要求フレーム，情報処理学会論文誌，Vol.28，No.4，pp.367-375，1987.
[大西 90]　大西淳（京大），阿草清滋（名大），大野豊（京都高度技術研究所）：要求フレームに基づいたソフトウェア要求仕様化技法，情報処理学会論文誌，Vol.31，No.2，pp.175-181，1990.
[大野治 95]　大野治，降旗由香里，津田道夫（日立）：データ部品方式による実践事例，bit 別冊ソフトウェアのモデル化と再利用（松本正雄（日本電気）編），pp.84-102，共立出版，1995.
[大野治 98]　大野治，降旗由香里（日立）：ソフトウェア標準化のためのプログラムパターンの作成，情報処理学会研究報告（ソフトウェア工学研究会）98-SE-121，pp.171-178，1998.
[大野治 2000]　大野治，降旗由香里，小室彦三，今城哲二，古宮誠一（日立）：

多次元部品方式によるソフトウェア開発の自動化――バッチプログラム用スケルトンの作成とその充分性――, 電子情報通信学会論文誌, Vol.J83-D-1, No.10, pp.1055-1069，2000.
[大野治 2001a] 大野治（日立）：多次元部品方式によるソフトウェア開発の自動化の研究，埼玉大学大学院理工学研究科博士学位論文，114 ページ，2001 年 3 月.
[大野治 2001b] 大野治, 小室彦三, 降旗由香里, 渡部淳一, 今城哲二（日立）：多次元部品方式によるソフトウェア開発の自動化――自動生成系の開発と評価――，電子情報通信学会論文誌，Vol.J84-D-1，No.9，pp.1372-1386，2001.
[大野豊 89] 大野豊（甲子園大），原田実（青山学院大）：自動プログラミングハンドブック，457 ページ，オーム社，1989.
[尾崎 86] 尾崎正治，松永義文，上林憲行（富士ゼロックス）：プロトタイプ Smaalltalk-80J～Smaalltalk-80 の日本語化についての検討～，情報処理学会第 32 回（昭和 61 年前期）全国大会，5F-1，pp.425-426，1986.
[尾関 94] 尾関哲（神戸高専），佐藤邦弘，太田健一，宮脇富士夫（姫路工大）：日本語プログラミング用エディタ評価のための識別子ベンチマークデータ，情報処理学会論文誌，Vol.35，No.9，pp.1949-1954，1994.
[尾関 95] 尾関哲（神戸高専），佐藤邦弘，太田健一，宮脇富士夫（姫路工大）：プログラム理解における日本語使用の効果，情報処理学会第 51 回（平成 7 年後期）全国大会，7M -3，pp.5-169～5-170，1995.
[尾関 99] 尾関哲（神戸高専），宮脇富士夫，太田健一，佐藤邦弘（姫路工大）：日本語ログラミング用エディタの開発，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.41-44，情報処理学会，1999.
[小高 88] 小高知宏，内山明彦（早大）：部品化・再利用向き言語“mCLD”によるプログラミング, 情報処理学会プログラミング言語研究会資料(16-1), pp.1-8, 1988-5-13.

か

[筧 87] 筧捷彦（早大）：基本プログラミング，244 ページ，岩波書店，1987.
[筧 89] 筧捷彦（早大）：Pascal と C への招待，223 ページ，岩波書店，1989.
[筧 95] 筧捷彦（早大），石畑清（明大），西田晴彦（NTT）：プログラム言語

最新情報－II 5. Ada-9X－大規模ソフトウェア向きの手続き型言語－, 情報処理, Vol.36, No.4, pp.304-314, 1995.
[風間 2000] 風間洋一（NTT）: Java プログラミング・ノート 国際化と日本語処理, 206 ページ, アスキー, 2000.
[片岡97]片岡裕（早大）: “国際化・多言語化の基礎と実際, bit, Vol.29, No.3, pp.48-56およびNo.4, pp.56-65, 共立出版, 1997.
[片桐 84] 片桐明（リギーコーポレーション）: FORTH, パソコン言語学（石田晴久監修）, pp.255-288, 1984.
[片桐 85] 片桐明(リギーコーポレーション):日本語プログラミング FORTH, bit 別冊 ワープロと日本語処理(石田晴久, 木村泉, 安田寿明編), pp.236-242, 1985.
[片桐 95a] 片桐明（リギーコーポレーション）: 日本語プログラム言語 Mind 基本文法（第３刷）, 287 ページ, リギーコーポレーション, 1995-6.
[片桐 95b] 片桐明（リギーコーポレーション）: 日本語プログラム言語 Mind PureMindPRO5.2 ユーザーズマニュアル（第７版３）, 248 ページ, リギーコーポレーション, 1995-11.
[加藤木 98] 加藤木和夫（日立プロセスコンピュータエンジニアリング）, 畠山正行（茨城大）: オブジェクト指向日本語一貫プログラミング環境, 情報処理学会研究報告(第 118 回ソフトウェア工学研究会), 98-SE-118, pp.15-22, 1998 年 3 月 10 日.
[加藤木 99] 加藤木和夫（日立プロセスコンピュータエンジニアリング）, 畠山正行（茨城大）: オブジェクト指向日本語一貫プログラミング環境, 情報処理学会論文誌, Vol.40, No.7, pp.3016-3030, 1999.
[兼宗 2000] 兼宗進, 久野靖（筑波大）: 学校教育用オブジェクト指向言語/環境の構想について, 情報処理学会シンポジウムシリーズ（情報教育シンポジウム論文集）, Vol.2000, No.9, pp.79-82, 情報処理学会, 2000-7-29～31.
[兼宗 2001a] 兼宗進, 久野靖（筑波大）: 学校教育用オブジェクト指向言語“ Dolittle ”の提案, 第 42 回プログラムシンポジウム, pp.16-20, 情報処理学会, 2001.
[兼宗 2001b] 兼宗進（筑波大）, 御手洗理英（筑波大／アーマット）, 中谷多哉子（エス・ラグーン）, 福井眞吾, 久野靖（筑波大）: 学校教育用オブジェク

ト指向言語「ドリトル」の設計と実装, 情報処理学会論文誌：プログラミング, Vol.42, No.SIG11(PRO12), pp.78-90, 2001.
[加野島 89] 加野島英雄, 大網呼人：日本語でプログラムを書く方法, 143 ページ, 翔泳社, 1989.
[神田 78] 神田泰典, 杉本正勝（富士通）：ソフトウェア工学における日本語の役割, 情報処理学会ソフトウェア工学研究会資料(5-1), pp.1-9, 1978-1-26.
[機シ振協 88] 機械システム振興協会（委託先：三菱総研）：21 世紀における新社会システムに関する調査報告書―日本語プログラミングの調査―（62-R-6(2)), 168 ページ, 1988.
[木下 85] 木下恂（東芝）：標準プログラム言語における日本語処理, 情報処理, Vol.26, No.3, pp.226-232, 1985.
[木村明 88] 木村明（マイクロソフトウェア・アソシエイツ), 片桐明（リギーコーポレーション）：日本語プログラミング言語「Mind」についてその概要と, 日本語プログラミングの実用性, 情報処理学会プログラミング言語研究会資料 (16-4), pp.25-32, 1985-5-13.
[木村俊一 87] 木村俊一（長野信連）：日本語プログラミング, 昭和 62 年度人口知能全国大会（第 1 回）9-5, pp.465-468, 1987.
[共立辞典 94] 日本ユニシス編：共立総合コンピュータ辞典第 4 版, pp.740-741, 共立出版, 1994.
[清兼 98] 清兼義弘, 末廣陽一編著（日本 DEC）：国際化プログラミン－I18N ハンドブック, オーム社, 480 ページ, 1998.
[草薙 85] 草薙裕（筑波大）：自然言語とコンピュータ言語, 216 ページ, 講談社サイエンティフィク, 1985.
[黒川清 93] 黒川清, 中川優, 関根純（NTT）：複合語解析技術を用いたデータ項目名称の標準化手法, 情報処理学会論文誌, Vol.34, No.3, pp.447-456, 1993.
[黒川利明 89] 黒川利明（日本 IBM), 湯淺太一（豊橋技科大), 橋本ユキ子（日電), 梅村恭司（NTT), 伊藤貴康（東北大）：LISP における国際文字処理方式に関する技術的諸問題, 情報処理学会記号処理研究会資料（51-1), pp.1-8 , 1989-6-2.
[黒川利明 90] 黒川利明（日本 IBM）：作品としてのプログラム, 191 ページ,

岩波書店，1990.
[黒川利明 98]　黒川利明（日本 IBM）：プログラム言語の仕組み，170 ページ，朝倉書店，1998.
[黒田 88]　黒田幸明（NTT）：FORTRAN における日本語化の現状と今後の方向，情報処理学会プログラミング言語研究会資料（16-8），pp. 49-52，1988-5-13
[小谷 87]　小谷善行，末永富美代，高田正之（東京農工大）：日本語 Prolog の利用者インターフェース，情報処理学会第 34 回（昭和 62 年前期）全国大会，3U-1，pp.743-744，1987.
[小谷 93]　小谷善行（東京農工大）：日本語に基づく論理プログラム表現，情報処理学会論文誌，Vol.34，No.5，pp.973-984，1993.
[古宮 90]　古宮誠一（情報処理振興事業協会）：事務処理ソフトウェア開発用簡易言語（第 4 世代言語）の現状と分析，情報処理学会，Vol.31，No.9，pp.1257-1269，1990.
[近藤 64]　近藤頌子，原田賢一，土居範久（慶大）：カナ文字 Fortran，情報処理，Vol.5，No.4，pp.222-223，1964.
[紺屋 87]　紺屋真一（富士通）：BASIC 言語における国語機能のサポート，情報処理学会第 32 回（昭和 62 年前期）全国大会，3V-2，pp.827-828，1987.

さ

[斎藤 91]　斎藤伸隆（キヤノンソフトウェア）：CANO-AID の開発から商品化まで，ソフトウェア生産工学ハンドブック（技術士ソフトウェア研究会編），pp.850-858，フジ・テクノシステム，1991.
[阪井 90]　阪井隆晴，楢府坦，今津幸雄（富士通）：アプリケーション開発支援ツールとディクショナリシステム，FUJITSU，Vol.41，No.5，pp.403-412，1990.
[酒井 84]　酒井紘昭（富士通）編著：簡易言語とエンドユーザズ言語，昭晃堂，167 ページ，1984.
[榊原 96]　榊原毅，乾伸雄，野瀬隆，小谷善行，西村恕彦（東京農工大）：日本語的な文法をもつ LOGO の文法設計，情報処理学会第 52 回（平成 8 年前期）全国大会，2N-1，pp.5-1～5-2，1996.
[島崎 81]　島崎真昭（京大）：日本語 Pascal：パスカル，情報処理学会第 23 回

（昭和 56 年後期）全国大会，1H-1，pp.215-216，1981.
[菅野 83] 菅野淳，本田邦夫，岡村嘉巳，上田謙一（松下技研）：自然言語の利用でマン・マシーン・インターフェースを改善する日本語 AFL，情報処理学会コンピュータにおけるヒューマン・インターフェースシンポジウム報告集，pp.53-65，1983-7-22.
[菅野 85] 菅野淳，上田謙一（松下技研）：日本語 AFL，bit 別冊 ワープロと日本語処理（石田晴久，木村泉，安田寿明編），pp.243-251，1985.
[杉尾 84] 杉尾俊之，武内惇，椎野努：日本語をベースにした仕様記述言語 NBSG における仕様記述について，情報処理学会ソフトウェア工学研究会資料（34-13），pp.73-84，1984-2-10.
[杉山 84] 杉山健司，秋山幸司，亀田雅之，竹之内顕文（富士通研），対話型自然言語プログラミングシステムの試作，電気通信学会論文誌，Vol.J67-D，No.3，pp.297-304，1984.
[鈴木茂夫 89] 鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡（東京農工大）：OS／omicron における日本語プログラミング環境，情報処理学会論文誌，Vol.30，No.1，pp.2-11，1989.
[鈴木孝則 83] 鈴木孝則（国際データ機器）：日本語プログラミング言語『和漢』，情報処理学会マイクロコンピュータ研究会資料（29-2），10 ページ，1983-11-28.
[鈴木弘 99] 鈴木弘（都立航空工業高専／慶大），中鉢欣秀，大岩元（慶大）：日本人のための初心者向けプログラミング教育用言語の試作，情報処理学会第 59 回（平成 11 年前期）全国大会，2X-06，1999.
[鈴木文音 99] 鈴木文音，大坪稔房，勝瑞雅也，湯浦克彦，津田道夫，宮崎肇之，降旗由香里，小室彦三，大野治（日立）：Java によるシステム開発設計と COBOL による現行系との比較評価，情報処理学会シンポジウムシリーズ（オブジェクト指向シンポジウム'99），Vol.99，No.9，pp.87-90，1999 年 7 月．
[鈴木康彦 99] 鈴木康彦，野口健一郎，後藤英一（神奈川大）：Java をターゲットにした自国語プログラミングの実験，情報処理学会第 58 回（平成 11 年前期）全国大会，5M-08，1999.
[関根 93] 関根純，川下満，町原宏毅，中川優（NTT）：体系的な DB 構築のための用語辞書を用いたデータ標準化手法, 情報処理学会論文誌, Vol.34, No.3,

pp.457-467，1993.
[芹沢 94] 芹沢浩（都立青鳥養護学校）：日本語 Logo 入門，183 ページ，森北出版，1994.

た

[高橋 88] 高橋延匡：研究プロジェクト総説：OS/omicron の開発，情報処理学会オペレーティングシステム研究会（39-5），14 ページ，1988-6-17.
[高橋 89] 高橋延匡（東京農工大）：日本語プログラミング環境，情報処理，Vol.30，No.4，pp.363-372，1989.
[玉井 88] 玉井哲雄，小久保岩生，牧野京子（三菱総研）：日本語プログラミングに関する実験と考察，日本ソフトウェア科学会第 5 回大会論文集，B2-1，pp.81-84，1988.
[玉井 90] Tetsuo Tamai（筑波大）：On Japanese-based Programming, Journal of Information Processing, Vol.13, No.1, pp.49-56, 1990.
[玉井 98] 玉井哲雄（東大）：コンピュータの言語，新・知の技法（小林康夫，船曳建夫（東大）編），pp.151-163，東京大学出版会，1998.
[千吉良 87] 千吉良英毅，永松祐嗣，小林正和（日立）：システム仕様書の再利用によるソフトウェアの開発技法(ICAS-REUSE)，日立評論，Vol.69, No.3, pp.249-254，1987.
[中所 98] 中所武司（明大）：ソフトウェア工学，196 ページ，朝倉書店，1998.
[中鉢 97] 中鉢欣秀，大岩元（慶大）：Java ヴァーチャルマシンをターゲットとした日本語オブジェクト指向言語の開発，情報処理学会プログラミング研究会（13-6），pp.31-38，1997-5-21.
[中鉢 98] 中鉢欣秀（慶大）：日本語指向のプログラミング，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.7-9，情報処理学会，1999.
[津田 92] Michio Tsuda, Yosuke Morioka, Masato Takadachi Mayumi Takahashi : Productivity Analysis of Software Development with an Integrated CASE Tool, International Conference on Software Engineering (ICSE92), pp.49-58, ACM, 1992.
[電子協 84] 言語標準化専門委員会：プログラミング用言語の標準化に関する調査―日本語 FORTRAN―, 59-C-483, pp.1-109, 日本電子工業振興協会, 1984.

[電子協 85a] 言語標準化専門委員会：プログラミング用言語の標準化に関する調査―日本語 COBOL―，60-C-506，202 ページ，日本電子工業振興協会，1985.
[電子協 85b] 言語標準化専門委員会：JEIDA 規格 電子計算機プログラミング言語 日本語 FORTRAN，JEIDA-42-1985，60-C-526，126 ページ，日本電子工業振興協会，1985.
[電子協 86] 言語標準化専門委員会：プログラミング用言語の標準化に関する調査―日本語 COBOL 他―，61-C-533，日本電子工業振興協会，1986.
[土居 87] 土居範久（慶大），筧捷彦（早大）：プログラミングの考え方，171 ページ，岩波書店，1987.
[土居 91] 土居範久：基礎 C 言語，351 ページ，岩波書店，1991.
[床分 86] 床分眞一，今城哲二（日立），花田良平（日立ソフトウェアエンジニアリング）：COBOL 言語における日本語機能，情報処理学会第 32 回（昭和 61 年前期）全国大会，5F-2，pp.427-428，1986.
[床分 88] 床分眞一，今城哲二（日立）：COBOL における日本語機能の現状と今後の方向，情報処理学会プログラミング言語研究会資料（16-9），pp.53-56，1988-5-13.

な

[中川 92a] 中川正樹，玉木祐二，早川栄一，曽谷俊男（東京農工大）：母国語プログラミングへの方式，実践とその効果，情報処理学会ソフトウェア工学研究会資料(85-2)，pp.7-14，1992-5-21.
[中川 92b] 中川正樹，早川栄一，玉木祐二（東京農工大）：日本語プログラムの可読性の評価と検討，情報処理学会ヒューマンインターフェース研究会資料(43-1)，pp.1-8，1992-7-9.
[中川 94a] 中川正樹，早川栄一（東京農工大），玉木祐二（東芝），曽谷俊男（日本 IBM）：日本語プログラミングの実践とその効果，情報処理学会論文誌，Vol.35，No.10，pp.2170-2179，1994.
[中川 94b] 中川正樹，早川栄一，並木美太郎，高橋延匡（東京農工大）：母語プログラミングの理念，実現，実践とその効果，電子情報通信学会論文誌 D-I，Vol.J77-D-I，No.5，pp.364-374，1994.

[中田 87] Nakata（筑波大）：Requirements for Character Handling in Programming Languages，ISO SC22 会議資料，1987. （[日本語委 88]に転載されている．SIGPLAN NOTICES Vol.23 (1988)の No.1(pp.5)と No.6 (pp.17-18)の Letter 欄にも関連記事が載っている．）
[中田 88] 中田育夫（筑波大）：プログラミング言語における日本語化の現状と今後の方向（総論），情報処理学会プログラミング言語研究会資料（16-6），pp.41-44，1988-5-13.
[中田 95] 中田育男（筑波大）：コンパイラ，193 ページ，オーム社，1995.
[中谷 2001] 中谷多哉子（エス・ラグーン），兼宗進（筑波大），御手洗理英（アーマット），福井眞吾，久野靖（筑波大）：オブジェクトストーム：プログラミング言語と情報教育の新しい関係，オブジェクト指向最前線 2001（情報処理学会 oo2001 シンポジウム），pp.57-64，近代科学社，2001 年 9 月.
[中村 85] 中村昭次，久野靖，木村泉（東工大）：日本語 CLU システムの試作，情報処理学会第 30 回（昭和 60 年前期）全国大会，1R-7，pp.461-462，1985.
[那野 83] 那野比古：要説日本語 AFL，189 ページ，東京ブック，1983.
[並木 92] 並木美太郎，早川栄一，下村秀樹，田中泰夫，中川正樹，高橋延匡（東京農工大）：言語 C コンパイラのマルチバイト化の実現方式，情報処理学会論文誌，Vol.33，No.11，pp.1331-1340，1992.
[西尾 93] 西尾高典，秋山美登，今城哲二（日立）：アプリケーションの分散開発を実現する第 4 世代言語，日立評論，Vol.75，No.9，pp.605-610，1993-9.
[西垣 2001] 西垣通，ジョナサン・ルイス：インターネットで日本語はどうなるか，245 ページ，岩波書店，2001.
[西之園 93] 西之園春夫（研究代表者：京都教育大学）：情報教育のための日本語プログラミング言語 Mind の実用化に関する研究，平成 4 年度科学研究費補助金（試験研究（B）(1)）研究成果報告書（課題番号：03558030），1993，約 400 ページ.
[西村 67] 西村恕彦（通産省電気試験所）：日本語と COBOL，情報処理，Vol.８，No.3，pp.157-160，1967.
[西村 86] 西村恕彦（東京農工大）：FORTRAN 日本語処理の JIS 原案，情報処理学会記号処理研究会（36-6）/プログラミング言語研究会（5-6）共催資料，pp.1-4，1986-3-10.

[西村 93]　西村恕彦（東京農工大），植村俊亮（奈良先端大）：入門 COBOL（新版），236 ページ，オーム社，1993.
[日経データプロ 92]　ケース・スタディ　丸紅　HITAC 向け CASE ツールで IBM 用ソフトの開発も可能に, 日経データプロ・ソフト, NS2-501-701, 1992-3.
[日経データプロ 94]　製品レポート CANO-AIDⅢ, 日経データプロ・ソフト, NS2-544-301，1994-2.
[日経データプロ 97a]　製品レポート CASET，日経データプロ・ソフト, NS2-544-701，1997-3.
[日経データプロ 97b]　製品レポート YPS 日経データプロ・ソフト, NS2-544-721，1997-3.
[日本語委 88]　日本語機能専門委員会：情報技術における日本語機能の標準化に関する調査研究報告書，77 ページ，情報処理学会情報規格調査会／日本規格協会情報技術標準化推進センター，1988-3.　（1989 年と 1990 年にもこの委員会の報告書が発行されている.）
[日本語 FORT86]　日本語 FORTRAN JIS 原案作成委員会：電子計算機プログラミング言語 日本語 FORTRAN JIS 原案，1986.

は

[葉木 84]　葉木洋一，今城哲二，津田道夫，仁平博三（日立）：システム開発支援ソフトウェア“ EAGLE ”，日立評論，Vol.66，No.3，pp.19-24，1984.
[葉木 86]　葉木洋一（日立), 北尾修治（日立ソフトウェアエンジニアリング），千吉良英毅，津田道夫，大野治，仁平博三（日立）：システム開発支援ソフトウェア“ EAGLE ”－EAGLE 拡張版 EAGLE2（日立），日立評論，Vol.68，No.5，pp.373-378，1986.
[長谷部 87]　長谷部紀元（図書館大）：プログラミング言語における日本語サポート方式―多言語サポートの国際標準化の脈絡から―, 情報処理学会プログラミング研究会資料（12-4），10 ページ，1987-7-24.
[畠山 2000]　畠山正行（茨城大），加藤木和夫（日立プロセスコンピュータエンジニアリング），石井義之（茨城大）：オブジェクト指向記述日本語 OODJ とその記述環境，情報処理学会論文誌，Vol.41，No.9，pp.2567-2581，2000.
[原田實郎 82]　原田實郎，仁平博三（日立），他：オンラインデータベース向

き業務プログラム開発維持支援システム「漢字 CORAL」, 日立評論, Vol.64 , No.5, pp.351-354, 1982.
[原田実 93] 原田実, 中村義幸（青山学院大）：プログラムの構造と論理の自動設計システム EOS/M, 情報処理学会論文誌, Vol.34, No.9, pp.2013-2024, 1993.
[ハンドブック 91] システムズエンジニアリングハンドブック, pp.407-409, オーム社, 1991.
[日立 95] 日立製作所：SEWB3 第 4 世代言語文法マニュアル（第 3 版), 3000-7-475-20, p.533, 1995.
[標準化フォーラム 88] 情報技術標準化フォーラム「日本語処理の統一的取り扱い」講演資料, 76 ページ, 情報処理学会情報規格調査会, 1988-12-7.
[標準化フォーラム 90] 情報技術標準化フォーラム「プログラム言語とデータベース」講演資料, 111 ページ, 情報処理学会情報規格調査会, 1990-1-23.
[平尾 86] 平尾隆行（日本 IBM）：日本語 APL 入門, 222 ページ, オーム社, 1986.
[平賀 89] 平賀正樹（富士通), 日本語プログラム言語 Mind, コンピュータソフトウェア, Vol.6, No.2, pp.170-178, 1989.
[平川 86] 平川知親, 中村康一, 谷部幸男（日立), 戸金旗一（日立コンピュータコンサルタント)：自動ドキュメンテーション支援システム“ADCAS”－金融アプリケーションパッケージへの適用例－, 日立評論, Vol.68, No.5, pp.369-372, 1986.
[平林 88] 平林雅英：C 言語の日本語化とその効果, 情報処理学会プログラミング言語研究会資料 (16-2), pp.9-16, 1988-5-13.
[福山 85] 福山峻一, 小林吉純, 伊集院正（NTT), 木下恂, 落合正俊（東芝)：Ada 日本語処理機能の実現法, 情報処理学会第 30 回（昭和 60 年前期）全国大会, 1R-6, pp.459-460, 1985.
[藤崎 86] 藤崎哲之助（日本 IBM)：日本語によるデータベース照会, 日本語情報処理（高橋延匡（東京農工大）編), pp.205-252, 1986.
[藤原 2001] 藤原晃, 楠本真二, 井上克郎（阪大), 大坪稔房, 湯浦克彦（日立)：複雑度と機能量に基づくアプリケーションフレームワークの実験的評価, オブジェト指向最前線 2001 情報処理学会 oo2001 シンポジウム, pp.85-90, 近

代科学社，2001 年 9 月.
[プロシン 99]　秋のプログラム・シンポジウム「日本のプログラミング」報告集 1998 年 9 月 16 日～18 日，情報処理学会，1999.
[本位田 98]　本位田真一，大須賀昭彦（東芝）：オブジェクト指向からエージェント指向へ，ソフトバンク，1998.

ま

[松尾 86]　松尾篤弥，牛島和夫（九大）：Ada における日本語テキスト処理パッケージの構築とその使用，情報処理学会記号処理研究会（36-2）／プログラミング言語研究会（5-2）共催資料，7 ページ，1986-3-10.
[松野下 96]　松野下博司，園田雅紀，河合敦夫，椎野努（三重大），武内惇（日大）：仕様書記述制限日本語に対する記述支援系の開発，情報処理学会第 53 回（平成 8 年後期）全国大会，2D-6，pp.1-219～1-220，1996.
[三上 88]　三上次郎, 東村昭男(富士通)：高生産ツール YPS, FUJITSU, Vol.39, No.1，pp.29-35，1988.
[水谷 85]　水谷静夫（東京女子大）：国語屋から見た総論，bit 別冊 ワープロと日本語処理（石田晴久（東大），木村泉（東工大），安田寿明（東京電機大）編），共立出版，pp.19-30，1985.
[水谷 86]　水谷静夫：次第書き言語《小朱唇》の設計思想，情報処理学会記号処理研究会（36-3）／プログラミング言語研究会（5-3）共催資料，pp.1-8，1986-3-10.
[南 83]　南俊郎（九大），杉尾俊之，武内惇，椎野努（沖電気）：日本語をベースとした仕様記述言語 NBSG，情報処理学会ソフトウェア工学研究会資料（30-2），7 ページ，1983-6-29.
[宮脇 93]　宮脇富士夫，佐藤邦弘（姫路工大）：日本語プログラミング環境のための開発と評価,「利用者指向の情報システム」シンポジウム, pp.125-134, 1993-11.
[宮脇 94]　宮脇富士夫（姫路工大），尾関哲（神戸高専），太田健一，佐藤邦弘（姫路工大）：日本語プログラミング言語（日本語 C++）の開発，姫路工業大学工学部研究報告，No.47，pp.70-82，1994.
[宮脇 99]　宮脇富士夫（姫路工大），尾関哲（神戸高専），太田健一，佐藤邦弘

（姫路工大）：日本語プログラミング環境の開発例，秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日~18 日，pp.33-40，情報処理学会，1999.
[村上 87] 村上憲稔，宮成功，富田嘉文，野沢隆，林義雄（富士通）：YACⅡエディタ，構造エディタ，pp.135-146，共立出版，1987.
[元吉 87] 元吉文夫(電総研)：Common Lisp における日本語処理方式の提案，情報処理学会記号処理研究会資料（40-6），8 ページ，1987-1-13.
[元吉 88] 元吉文夫（電総研）：Common Lisp における日本語化，情報処理学会プログラミング言語研究会資料（16-12），pp.65-68，1988-5-13
[森本 84] 森本陽二郎，中山康子，南川忠利（東芝）：パソコン用オブジェクト指向型言語，情報処理学会第 29 回（昭和 59 年後期）全国大会，2L-1，pp.1549-1550，1984.
[諸橋 76] 諸橋正幸, 藤崎哲之助, 鷹尾洋一, 間下浩之, 渋谷政昭（日本 IBM）：「ヤチマタ」における擬似日本語，情報処理学会計算言語学(CL)研究会資料（8-2），pp.1-10，1976-12-17.

や

[矢島 89] 矢島廣，長谷川栄，風間順一，山口康隆，高崎繁夫（日立），加藤礼吉（日立ソフトウェアエンンジニアリング）：オペレーティングシステム VOSK の基本制御方式，日立評論，Vol.71，No.11，pp1111-1118，1989.
[安本 92] 安本太一（愛知教育大），湯淺太一（豊橋技科大）：Kyoto Common Lisp の日本語文字処理機能の実現とその評価，コンピュータソフトウェア，Vol.9，No.5，pp.391-402，1992.
[山之上 87] 山之上卓，安在弘幸，吉田将（九大），杉尾俊之，武内惇，椎野努（沖電気）：言語処理系の生成系 MYLANG による NSBG/PD プリコンパイラの試作，情報処理学会論文誌，Vol.28，No.1，pp.64-73，1987.
[湯浦 2002] 湯浦克彦，大坪稔房，団野博文（日立），石井義明（日立システムアンドサービス），古澤憲一（イーシー・ワン），桐越信一，鈴木文音（日立）：EJB コンポーネントによる Web システム構築技法，380 ページ，ソフト・リサーチ・センター，2002 年 3 月.
[吉田和幸 85] 吉田和幸（大分大），牛島和夫（九大）：SNOBOL4 既存処理系への日本語テキスト処理機能の追加, コンピュータソフトウェア, Vol.2, No.3,

pp.518-528，1985.

[吉田正和 93] 吉田正和：日本語 C プログラミング，インターフェース，1993 年 4 月号，pp.242-244，CQ 出版社.
[吉野 91] 吉野松樹，谷口恵子，西山勲（日立）：分散形第 4 世代言語 EAGLE/4GL, 情報処理学会第 42 回（平成 3 年前期）全国大会, 5S-6, pp.5-345～5-346，1991.
[吉野 93] 吉野松樹，田村和敏（日立），稲益良夫（日立ソフトウェアエンジニアリング）：ソフトウェア開発支援ツール“ SEWB3，EAGLE/4GL ”の機能と特長，日立評論，Vol.75，No.11，pp.727-734，1993-11.

わ

[脇坂 89] 脇坂隆則（日立ソフトウェアエンジニアリング），秋山美登，石井武夫，中田恵都子，今城哲二（日立）：中小形システム VOS K(4)－第 4 世代言語 EAGLE/4GL－, 情報処理学会第 38 回（平成元年前期）全国大会, pp.835-836, 1989.
[和田英一 88] 和田英一（東大）：プログラミング言語で使う文字コード，情報処理学会プログラミング言語研究会資料（16-7），pp.45-48，1988-5-13.
[和田孝 88] 和田孝，土田賢省，杉山高弘，阪田全弘，富田兼一，宮下洋一（日電）：プログラミング自動生成のための日本語仕様記述言語，情報処理学会プログラミング言語研究会資料（16-5），pp.33-40，1988-5-13.
[和田英穂 91] 和田英穂（富士通）：プログラム言語の将来，コンピュータの事典第 2 版，pp.430-432，朝倉書店，1991.
[和田英穂 95] 和田英穂（富士通）：プログラム言語最新情報－Ⅱ 4．新しい Fortran－Fortran 90－，情報処理，Vol.36，No.4，pp.297-303，1995.
[渡辺勝正 80] 渡辺勝正，都司達夫（福井大）：日本語プログラム言語の開発にむけて, 情報処理学会第 21 回（昭和 55 年）全国大会, 5I-1-1, pp.1011-1012, 1980.
[渡辺坦 89] 渡辺坦（日立）：パタン照合型プログラミング方式の提案，情報処理学会プログラム言語研究会報告（20-1），pp.1-9，1989-2-10.

アルファベット

[COBOL 研 67] 情報処理学会 COBOL 研究会：国産の COBOL コンパイラ，情報処理，Vol. 8，No.3，pp.140-144，1967.

[CODASYL94] CODASYL COBOL Committee：Journal of Development 1993, ISDN 0-9621665-1-4, 1994.

[ISOAda95] ISO/IEC 8652: 1995 Information technology – Programming language – Ada Edition: 2 (monolingal).

[ISOC++98] ISO/IEC 14482:1998 Information technology – Programming language – C++. （対応する JIS は 2002 年に発行予定である.）

[ISOCOBOL2001] ISO/IEC FCD1989：2001 Information technology － Programming Language － COBOL，869 ページ，2001-01-15. （2002 年春に DIS（国際推薦規格），2002 年末に IS（国際規格）が発行され，IS に対応する JIS は 2003 年度に発行される見込みである.）

[ISOProlog95] ISO/IEC 13211-1: 1995 Information technology – Programming language – Prolog Part 1：General core.

[JISC93] JIS X 3010-1993 プログラム言語 C. （対応する国際規格番号は ISO/IEC 9899:1990 である.）

[JISC96] JIS X 3010-1996 プログラム言語 C (追補 1). （対応する国際規格は Amendment 1:1995 to ISO/IEC 9899: 1990 C Integrity である.）

[JISFORT94] JIS X 3001:1994 プログラム言語 Fortran. （対応する国際規格番号は ISO/IEC 1539:1991 である.）

[JISISLISP98] JIS X 3012:1998 プログラム言語 ISLISP. （対応する国際規格番号は ISO/IEC 13816:1997 である.）

[JISMUMPS95] JIS X 3011:1995 プログラム言語 MUMPS （対応する国際規格番号は ISO/IEC 11756:1992 である.）

[JISPOSIX94] JIS X 3030-1994 移植可能なオペレーティングシステムのインターフェース(POSIX) 第 1 部 応用プログラム向けのインターフェース(API) [プログラム言語 C]. （対応する国際規格番号は ISO/IEC 9945-1:1990 である.）

[JISPOSIX96] JIS 原案 移植可能なオペレーティングシステムのインター

フェース(POSIX) 第 2 部　シェルとユーティリティ，1996.　（対応する国際規格番号は ISO/IEC9945-2:1993 である.）
[JISSQL95]　JIS X3005:1995 データベース言語 SQL.　（対応する国際規格番号は ISO/IEC 9075:1992 である.）
[JISUCS95]　JIS X 0221:1995　国際符号化文字集合（UCS）－第一部 体系及び基本多言語面, 日本規格協会, 1995.　（対応する国際規格番号はISO/IEC 10646-1 1993である．2000年にJISは改訂されている.）
[JTC1TSG91]　ISO/IEC JTC1 TSG-1 on Standards necessary to define Interface for Application Portability (IAP) FINAL REPORT, ISO/IEC JTC1 N1335, April 1991.　（これは[SSI91]に転載されている.）
[Martin85]　Martin, J：Forth-Generation Languages, Vol.1, 2, and 3, Prentice-Hall, 1985.
[NHK88]　NHK 取材班編：日本語プログラミングの試み, “ 世界の中の日本：コンピュータが日本を変える　第 2 巻（コンピュータは言葉の壁を越えられるか）”, pp.210-223, 日本放送協会／角川書店, 1988.
[SSI91]　SSI専門委員会：システムインターフェイスの標準化に関する調査研究－－OSインターフェース標準化の検討－－報告書, 情報処理学会情報規格調査会，1991.　（この委員会の報告書は1986年度から1990年度まで毎年発行されている.）
[TR 言語規格 2000]　TR X 0031:2000 プログラム言語規格策定の方針，41 ページ，日本規格協会，2000.（対応する国際規格技術報告番号は ISO/IEC TR 10176:1998 である.）
[TR 国際化 2000] TR X 0031:2000 プログラム言語規格策定の方針, 59 ページ, 日本規格協会，2000.（対応する国際規格技術報告番号は ISO/IEC TR 10176:1998 である.）
[Wirth79]　N,Wirth（片山卓也訳）：アルゴリズム＋データ構造＝プログラム，414 ページ，日本コンピュータ協会，1979.

研究業績一覧

1. 学会論文誌（査読あり）

[1]　三宅立記，今城哲二，佐藤忍，井藤敏之，横塚大典，辻畑好秀，植村俊亮：Web 対応事務処理スクリプト言語「COBOL スクリプト」，情報処理学会論文誌プログラミング，Vol.41，No. SIG 2（PRO 6），pp.89-101，2000.　（第 2 章に関連）

[2]　大野治，降旗由香里，小室彦三，今城哲二：多次元部品方式によるソフトウェア開発の自動化――プログラム自動生成方式――，電子情報通信学会論文誌，第 J84-D-1 巻，第 9 号，pp.1372-1386，2001 年 9 月.　（第 3 章に関連）

[3]　大野治，降旗由香里，小室彦三，今城哲二，古宮誠一：多次元部品方式によるソフトウェア開発の自動化――バッチプログラム用スケルトンの作成とその十分性――, 電子情報通信学会論文誌, Vol.J83-D-1, No.10, pp.1055-1069, 2000 年 10 月.　（第 3 章に関連）

2. 著書

[1]　OMG　Japan　SIG 翻訳委員会 UML 作業部会訳：UML 仕様書，アスキー出版，800 ページ，2001 年 10 月.＜UML 作業部会の 18 人で分担翻訳．今城は部会主査＞

[2]　今城哲二編, 細島一司著:標準 COBOL プログラミング, カットシステム, 472 ページ，1999 年 11 月.

[3]　金子敬一，今城哲二，中村英夫：入門計算機ソフトウェア，朝倉書店，208 ページ，2000 年 4 月.　（第 4 章と付録 B に関連）

[4]　今城哲二編，吉野松樹，銀林純，成田雅彦，石田厚子，小池博，増石哲也，高橋まゆみ，湯浦克彦，団野博文，橋本惠二，大槻繁，大谷真共著：ビジネスオブジェクト入門，ソフト・リサーチ・センター，341 ページ，2000 年 7 月.

[5]　岩波情報科学事典，岩波書店，1990 年 5 月.＜今城は付録の COBOL と RPG のプログラム例を分担執筆＞

[6]　今城哲二：明解 COBOL，サイエンス社，213 ページ，1988 年 9 月.　（第 4 章に関連）

3. 国際会議での発表論文（査読あり）

[1] Tetsuji Imajo, Tatsuki Miyake, Shinobu Sato, Toshiyuki Ito, Daisuke Yokotsuka, Yoshihide Tsujihata, and Shunsuke Uemura：COBOL Script：A Business-Oriented Scripting Language, Proc. of the fourth International Enterprise Distributed Object Computing Conference EDOC2000, pp.231-239, Makuhari, Japan, September 2000. （第 2 章に関連）

4. 学会の研究会・シンポジウムでの発表論文

＜2001 年＞

[1] 今城哲二，吉野松樹，大坪稔房，原田晃，大野治，植村俊亮：オンライン業務プログラムの環境独立処理方式，情報処理学会ソフトウェア工学研究会研究報告，2001-SE-135，pp.33-40，2001 年 11 月． （第 3 章に関連）

[2] 今城哲二，鈴木弘，大野治，植村俊亮：日本語プログラム言語“まほろば”の言語仕様，情報処理学会ソフトウェア工学研究会研究報告，2000-SE-130，pp.143-152，2001 年 3 月． （第 4 章に関連）

[3] 今城哲二, 大野治, 原田晃, 津田道夫：コンポーネント流通の現状と課題，情報処理学会ソフトウェア工学研究会ウィンタワークショップ・イン・金沢，情報処理学会シンポジウムシリーズ，Vol.2000，No.2，pp.89-90，2001 年 1 月.

＜2000 年＞

[4] 今城哲二，鈴木弘，大野治，植村俊亮：プログラム言語での多バイト符号系支援時の設計指針, 情報処理学第 29 回プログラミング研究会, 8 ページ, 2000 年 6 月 15 日. …情報処理学会論文誌：プログラミング, Vol.42, No. SIG 2（PRO 10），2001 年発行に発表概要掲載． （第 4 章に関連）

[5] 今城哲二，鈴木弘，大野治，植村俊亮：日本語プログラム言語”まほろば“の文法，情報処理学第 29 回プログラミング研究会，13 ページ，2000 年 6 月 15 日．…情報処理学会論文誌：プログラミング，Vol.42，No. SIG 2（PRO 10），2001 年発行に発表概要掲載． （第 4 章に関連）

[6] 鈴木弘，今城哲二，中鉢欣秀，大岩元：非分かち書き日本語プログラム言語のための字句解析, 情報処理学第 27 回プログラミング研究会, 7 ページ, 2000 年 1 月. …情報処理学会論文誌：プログラミング, Vol.41, No. SIG 2（PRO 8），

2000 年発行に発表概要掲載.　（第 4 章に関連）

＜1999 年＞

[7]　今城哲二，鈴木弘，大野治，植村俊亮：日本語プログラム言語 ”まほろば“ の文法と記述評価，情報処理学第 26 回プログラミング研究会，25 ページ，1999 年 10 月．…情報処理学会論文誌：プログラミング，Vol.41，No. SIG 2（PRO 7），2000 年発行に発表概要掲載.　（第 4 章と付録 B に関連）

[8]　今城哲二，大野治，津田道夫，高橋まゆみ：ビジネスオブジェクト近況，情報処理学会サマーワークショップ・イン・小樽論文集，情報処理学会シンポジウムシリーズ，Vol.99，No.11，pp.49-50，1999 年 9 月.

[9]　三宅立記，今城哲二，佐藤忍，井藤敏之，横塚大典，辻畑好秀，植村俊亮：Web 対応事務処理スクリプト言語「COBOL スクリプト」，第 25 回プログラミング研究会，12 ページ，1999 年 8 月 5 日．…情報処理学会論文誌：プログラミング，Vol.41，No. SIG 2（PRO 6），pp.89-101，2000 年 3 月に全文掲載.（第 2 章に関連）

[10]　今城哲二，鈴木弘，植村俊亮：日本語プログラム言語「まほろば」の構文設計，情報処理学会第 25 回プログラミング研究会，25 ページ，1999 年 8 月 5 日）．…情報処理学会論文誌：プログラミング，Vol.41，No. SIG 2（PRO 6），2000 年 3 月に発表概要掲載.　（第 4 章に関連）

[11]　今城哲二，佐藤敬幸，木戸彰夫，野田誠，高田正之，越田一郎，中村克彦，湯淺太一，植村俊亮：標準プログラム言語の国際化，情報処理学会第 23 回プログラミング研究会，17 ページ，1999 年 3 月 24 日．…情報処理学会論文誌：プログラミング，Vol.40，No. SIG 2（PRO 4），1999 年に発表概要掲載.（第 2 章に関連）

[12]　今城哲二，植村俊亮：日本語プログラム言語文献解題，情報処理学会第 23 回プログラミング研究会，10 ページ，1999 年 3 月 24 日．…情報処理学会論文誌：プログラミング，Vol.40，No. SIG 2（PRO 4），1999 年に発表概要掲載.　（付録 A に関連）

[13]　今城哲二：日本語擬似言語，情報処理学会ウィンタワークショップ・イン・高知論文集, 情報処理学会シンポジウムシリーズ, Vol.99, No.1, pp.51-52, 1999 年 1 月.　（第 4 章に関連）

[14]　今城哲二，秋葉俊介，鬼頭政義，辻畑好秀：日本からの ISO と OMG で

の標準化提案事例とその海外展開，電子情報通信学会第 5 回ソフトウェアグローバル競争力研究会，電子情報通信学会技術研究報告，SGC 99-17～20 第二分冊 〔ソフトウェアグローバル競争力〕，pp.1-10，1998 年 1 月． （第 2 章に関連）

[15] 今城哲二：日本語プログラム言語の設計，情報処理学会第 40 回プログラム・シンポジウム報告集，pp.9-16，1999 年 1 月． （第 4 章に関連）

＜1998 年＞

[16] 今城哲二：日本語プログラム文献ノート，情報処理学会秋のプログラム・シンポジウム報告集 1998 年 9 月 16 日～18 日，pp.9-16，1999． （付録 A に関連）

＜1988 年＞

[17] 床分眞一，今城哲二：COBOL における日本語機能の現状と今後の動向，情報処理学会第 16 回プログラム言語研究会，pp.53-56，1988 年 5 月． （第 2 章に関連）

5. 情報処理学会誌，日立評論などへの寄稿

[1] 今城哲二，横塚大典，床分眞一：21 世紀も COBOL だね～COBOL 生誕 40 周年トピックス～，bit，Vol.33，No.4，pp.63-73，2001 年 4 月． （第 2 章に関連）

[2] 今城哲二：国際標準活動トピックス「標準化プログラム言語の国際化」，情報技術標準，No.44，pp2-6，情報処理学会情報規格調査会，1999 年 2 月．（第 3 章に関連）

[3] 今城哲二：COBOL によるアプリケーション開発，日経データプロ・オープンプラットフォーム NT 版，1995 年 4 月号．

[4] 西尾高典，今城哲二，中原俊政，津田道夫：日立製作所のアプリケーション開発支援体系 “ CAPSDF ”，日立評論，Vol.75，No.11，pp.9-14，1993 年 11 月．

[5] 増位庄一，葉木洋一，梶山雄三，今城哲二：ソフトウェア生産技術の展望－ソフトウェアエンジニアリングの現状と将来－ ，日立評論，Vol.75，No.11，pp.4-8，1993 年 11 月．

[6] 西尾高典，秋山美登，今城哲二：アプリケーションの分散開発を実現する

第 4 世代言語，日立評論，Vol.75 ，No.9，pp.605-610，1993 年 9 月． （第 3 章に関連）
[7] Tetsuji Imajo, Yoshinori Akiyama, and Shuji Kitao：VOS K 4th Generation Language “EAGLE/4GL”, HITACHI REVIEW, Vol.39. No.5, pp.261-266, 1990. （第 3 章に関連）
[8] 今城哲二，秋山美登，北尾修治，里本健，脇坂隆則，大西俊治：VOSK 第 4 世代言語“ EAGLE/4GL ”，日立評論，Vol.71，No.11，pp.39-44，1989 年 11 月． （第 3 章に関連）
[9] 葉木洋一，今城哲二，津田道夫，仁平博三：システム開発支援ソフトウェア“ EAGLE ”，日立評論，Vol.66，No.3，pp.19-24，1984 年 3 月． （第 3 章に関連）
[10] 今城哲二:COBOL の標準化動向, 情報処理, Vol.24, No.9, pp.1062-1069, 1983 年 7 月.
[11] 今城哲二：COBOL，情報処理，Vol.22，No.6，pp.457-460，1981 年 6 月.

6. 講演（情報処理学会）

[1] 今城哲二：オブジェクト指向 COBOL の動向, 情報処理学会オブジェクト指向 ’96 シンポジウム．1996 年.
[2] 今城哲二：COBOL の日本語機能，情報技術標準化フォーラム「プログラム言語とデータベース」，1990 年 1 月 23 日． （第 3 章に関連）

7. 発表（情報処理学会全国大会）

[1] 脇坂隆則，秋山美登，石井武夫，中田恵都子，今城哲二：中小型システム VOS K(4)－第 4 世代言語 EAGLE/4GL－, 第 38 回全国大会, pp.835-836, 1989 年 3 月． （第 3 章に関連）
[2] 山田守一, 矢羽田正孝, 木原康博, 今城哲二, 小熊信一郎：COBOL COPY 文における接頭辞・接尾語の付加機能，第 36 回全国大会，1J-5，pp.829-830，1988 年 3 月.
[3] 三宅立記，花田良平，今城哲二：COBOL の拡張アドレスサポート方式，第 33 回全国大会，3D-5，p.395-396，1986 年 9 月.

[4] 今城哲二，高見沢正巳，森陽子，加藤豊：COBOL デバッガでの DB/DC 単体テスト機能，第 33 回全国大会，3D-7，pp.399-400，1986 年 9 月.
[5] 床分眞一，今城哲二，花田良平：COBOL 言語における日本語機能，第 32 回全国大会，5F-2，pp.427-428，1986 年 3 月.　（第 2 章に関連）
[6] 横山一郎，米田茂，田中和明，今城哲二：XDM（2）－DB 定義言語と操作言語－，第 32 回全国大会，4B-4， pp.841-842，1986 年 3 月.
[7] 武直行，上田恭雄，今城哲二：XDM（3）－DC 操作言語－，第 32 回全国大会，4B-5，pp.843-844，1986 年 3 月.
[8] 川村透，今城哲二，津田道夫：EDP 部門向け開発支援ツール“ EAGLE ”，第 29 回（昭和 59 年後期）全国大会，3R-1，1984.　（第 3 章に関連）
[9] 松本恵一，今城哲二，佐藤厚，河津秀子：超高性能電子計算機システム中間言語プロセッサの開発，第 13 回大会，16，pp.31-32，1972.

8. JIS/ISO などの標準規格及び標準化関連報告書　＜＞内は今城の役割

＜近々制定予定＞

[1] JIS　一般 ORB 間プロトコル（GIOP）及びインターネット ORB 間プロトコル（IIOP），2002 年発行予定．＜原案作成委員＞
[2] ISO/IEC 及び ANSI　COBOL，2002 年発行予定．＜原案作成委員＞（第 2 章に関連）

＜以下は発行済み＞

[3] EJB 日本語用語規約，EJB コンポーネントに関するコンソーシアムのホームページ　http://www.ejbcons.gr.jp，2001 年 7 月．＜この規約を作成した日本語 EJB 用語部会の主査＞
[4] JIS　TR X 0031:2000 プログラム言語規格策定の方針，41 ページ，日本規格協会，2000 年．＜原案作成委員＞　（第 2 章に関連）
[5] JIS　TR X 0030:2000 国際化の基本構造, 59 ページ, 日本規格協会, 2000 年．＜原案作成委員＞　（第 3 章に関連）
[6] JIS　COBOL，1992 年．＜原案作成委員長＞
[7] JIS　COBOL，1988 年．（原案作成委員）
[8] 情報処理技術における日本語機能の標準化に関する調査研究報告書，情報処理学会／日本規格協会，1988 年 3 月．＜この報告書を作成した日本語機能

専門委員会の委員（分担執筆）＞　（第 2 章に関連）
[9]　JIS　COBOL，1978 年．＜原案作成委員＞

9.　設計又は作品（主たる設計者として開発した製品）　年月日は完成日
[1]　日立製作所　（マルチターゲット向け）UNIX　システム開発支援ツール SEWB3，1993 年 3 月 20 日．　（第 3 章に関連）
[2]　日立製作所　UNIX　COBOL コンパイラ，1992 年 6 月 30 日．　（第 2 章に関連）
[3]　日立製作所　（中小計算機向け）第 4 世代言語 VOSK　EAGLE/4GL, 1988 年 12 月 20 日．　（第 3 章に関連）
[4]　日立製作所　（中小計算機向け）VOSK　COBOL コンパイラ，1988 年 12 月 20 日．　（第 2 章に関連）
[5]　日立製作所　（中型計算機向け）VOS１　COBOL コンパイラ，1986 年 3 月 31 日．　（第 2 章に関連）
[6]　日立製作所　（大型計算機向け）VOS3　COBOL コンパイラ，1985 年 8 月 31 日．　（第 2 章に関連）
[7]　日立製作所　（大型計算機向け）システム開発支援ツール　EAGLE，1983 年 9 月 30 日．　（第 3 章に関連）
[8]　日立製作所　超高性能電子計算機システム　中間言語プロセッサ，1972 年 9 月 30 日．