ソフトウェア工学 130−20
（2001．3．23）

日本語プログラム言語“まほろば”の言語仕様

今城哲二†,††　鈴木弘†††　大野治††††　植村俊亮††

約20年前からコンピュータでの漢字利用が普及し，標準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できる．予約語まで日本語にした本格的な日本語プログム言語も，分かち書きのレベルで，実用化されている．本論文では，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”の言語仕様について述べる．

Grammar of a Japanese-based Programming Language “Mahoroba”

Tetsuji Imajo †,††　Hiroshi Suzuki †††
Osamu Ohno ††††　Shunsuke Uemura ††

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 allow 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. This paper discusses a non-separated (No wakachigaki) version of Japanese-based programming language "Mahoroba". “Mahoroba” is a word of ancient Japan, and means a nice country. The paper desscribes it's design philosophy and syntax by experimental implementation.

1. はじめに

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，ソースプログラム自身も日本語化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[1]．

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[2)~4)]．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がより本格的な日本語プログラム言語の研究を進めている[5)~12)]．筆者は，日本語プログラム言語“まほろば”の研究・開発中で，すでに情報処理学会プログラムシンポジウム[6)]などで報告し，そのサブセット仕様のコンパイラ教育用のミニ言語“まほろば0”を大学用教科書で解説した[14)]．本論文では，“まほろば”の言語仕様に焦点をしぼり報告する．

† 日立製作所ソフトウェア事業部
Hitachi, Ltd. Software Division
†† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science,
Nara Institute of Science and Technology
††† 東京都立航空工業高等専門学校電子工学科
Tokyo Metropolitan College of Aeronautical Engineering, Department of Electronics Engineering
†††† 日立製作所公共システムグループ
Hitachi, Ltd. Public Systems Group

2. 言語仕様の設計方針

既存の日本語プログラム言語は分かち書きが多かったが，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は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” ことをを基本とする．

この基本方針を含め，“まほろば”の言語仕様の設計方針は次の４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．
(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関数としてサポートする．
(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．

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示す．次の例は，２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．

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

4. まほろば文法の記述法

まほろばの文法を 5～8 章に示す．この文法は次のメタ記号も用いて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．
　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．
… ：連続する自明な文字の省略を意味する．

5. 符号系と文字の種類

［符号系］

まほろばのソースプログラムは，次の２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．

［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＞

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る．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る．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文字について，まず英数字符号系と漢字符号系のまほろば文字集合を示

す．続いて，名前を構成する文字，句読点など区切りとなる記号，引用符と括弧，演算に用いる文字の4種類に分けて説明する．

[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＞｜＜英大文字＞
　｜＜英小文字＞｜＜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＞→
　　＋｜－｜＊｜／｜＝｜＞｜＜

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．

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．

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

[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．
(2) “々”など，仮名や漢字の繰返し記号が使える．
(3) 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．
(4) 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，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．
(5) 左かぎ括弧の“「”と，右かぎ括弧の“」”が使える．
(6) 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．

[名前を構成する文字]

＜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

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．

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．

JISなどの標準に存在するキリール文字とギリシャ文字は，まほろばでは漢字と同等として扱う．

[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．

［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＞→「｜」

［演算に用いる記号］
＜演算記号要素＞→　＋｜－｜×｜÷｜＝｜≠
　｜＞｜＜｜≧｜≦｜←｜→
＜正負符号＞→＋｜－

6. 語句の種類：名前，定数，予約語など

＜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＞｝）
　｜（[[＜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＞
＜空白列＞→＜空白＞｛＜空白＞｝
＜注釈＞→｜［＜計算機文字集合の文字＞］｜

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，１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，１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．

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”

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同じコードなので，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．

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．

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．

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．

注釈は，まほろばコンパイラの語句解析後は，あたかも１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．

それぞれの文字数の上限は，次のとおりとする．

- 名前と予約語：30 文字
- 数値定数： 30 文字（小括弧を除く）
- 単独の文字定数：１２０文字（左右の引用符を除く）

[行と空白の規則]

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．

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．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．これらの規則は，文字定数が行末で終わっていない場合にも適用される．

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．まほろばでも，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を除いた空白列のことである．

7. プログラムの構成と名前の定義

[構文規則の留意事項]

７章と８章の構文規則では，煩雑さをさけるため，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) 条件節の表現で，“のときは”や“になるまで”の前に“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．

ないのときは ⇒ ないときは
ないになるときは ⇒ なくなるときは
ないになるまで」 ⇒ なくなるまで
ないの間 ⇒ 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

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た．

[プログラムの構成]

＜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：＜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＞ | ＜ファイル名定義＞
　| ＜関数の引数の型定義＞}
＜処理部＞→【処理】{＜文＞} {＜関数＞}【終了】
＜仮引数＞→＜変数定義＞

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．関数は 3 つの部で構成される．宣言部や処理部は省略できる．

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．

名前部では，変数を定義する．変数の名前だけでなく，定数，ファイル，関数，データ型の名前を指定できる．

処理部では，文を0個以上記述する．

［定数名と変数名の定義］

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

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”．

定数名で，定数の直後の＜定数＞が数字定数のときは，間にコンマ“，”を書く．

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．ただし，関数群の前にくる共用の名前を書くための名前部の変数は，常に常駐とする．

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．レコード型については，後述する．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．

［データ型］

＜データ型＞→
　｜整数［＜精度＞］
　｜１０進数＜けた数＞けた
　　［，小数点以下＜けた数＞けた］［＜精度＞］
　　｜実数［＜精度＞］
　｜［＜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）
＜レコード型＞→レコード型，
　　「＜変数定義＞｛＜変数定義＞｝」
＜領域共用型＞→変数名：領域共用，
　　「＜変数定義＞｛＜変数定義＞｝」
＜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：データ型，
　　＜データ型＞．

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

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．10進数はSQLやCOBOLにもあり，事務処理向けの機能である．

けた数，文字数は，1以上の整数を指定する．

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．

［レコード型と領域共用型］

レコードは，複数の変数をまとめたものである．領域共用型は，レコード型の1つで，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．

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：文字列．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，「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．

［利用者定義のデータ型と列挙型，範囲型］

変数の値を限定する場合に，列挙型と範囲型を用いる．列挙型は，取る値に列挙名をつける．実行時に使われる値は，先頭から０．１．２…で，整数と

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．

［関数の引数の型指定］

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

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．

例：最大公約数：関数，引数は（正数，正数），もどり値は正数．

［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＞

配列全体を参照する場合には，【　】の中の添字は書かない．

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．

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５つがある．それらは，同一スコープ内で，次のグループの中で一意でなければならない．

(1) 関数名：　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，この２つは区別できる．
(2) 変数名，定数名，列挙型，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．
(3) データ型名

8.　文：処理の記述

［文の種類］

＜文＞→＜重文＞｜＜空文＞｜＜呼出し文＞｜＜復帰文＞
　　｜＜代入文＞｜＜２分岐文＞｜＜多分岐文＞
　　｜＜繰返し文＞｜＜脱出文＞｜＜入出力文＞
　　｜＜入出力条件文＞
＜重文＞→「＜文＞ {＜文＞}」
＜空文＞→何もしない．
＜呼出し文＞→＜関数参照＞．
＜復帰文＞→もどる［([＜式＞])］．

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．

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．

［代入文］

＜代入文＞→　＜変数参照＞＝＜式＞．
　　｜＜変数参照＞←＜式＞．
　　｜＜式＞→＜変数参照＞．
＜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＞→＋｜－

異なる型の代入はできない．ただし，数値どうしの代入は可能である．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．

式の中を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の構文解析を容易にするためである．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．

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

［２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］
＜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＞)}

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．

例：点数≧６０のときは，A→Z．それ以外は，B→Z．

［多分岐文］
＜多分岐文＞→
　　＜比較主体＞により分岐する．
　　「＜比較対象＞の場合，＜文＞
　　{＜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＞］

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：所属により多分岐する．
　　「総務の場合，「…」
　　営業の場合，「…」
　　その他の場合，「…」」

［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＞→
　　［最後が］＜条件＞になるまで
　｜［最後が］＜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．｜繰り返しの先頭にもどる．

脱出文は，繰返し文の中以外には書けない．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，制御は多分岐の最後の“」”には移らない．

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，入力レコード）＝終わりになるまで，
　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．….」」
この例の“ぬける”は，内側の繰返し文“配列要素が…,”に対し有効である．

［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．
- 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）
- 文字列の連結と分離
- 剰余演算
- 異なるデータ型の変換
- ファイルの入出力（レコード単位，フィールド（項目）単位）
- 数値の編集
- 日付や時刻の取得

9. 評価

まほろばを用いて，コンパイラ教育用ミニ言語“まほろば 0”のコンパイラ記述をし[13]，COBOL 学習書[14)～15)]のプログラム例題の記述実験を実施した．さらに，学生 5 人と社会人（SE）10 人にプログラミング教育を実施した．これらにより，まほろばでアルゴリズム記述と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．まほろばの実用化段階には実現する必要がある．

(1) レコード型の配列の初期値
(2) レコード型初期化文：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．
(3) 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．
(4) 長さが相手によって変わる標準定数名：SPACE，ZERO，HI-VALUE，LOW-VALUE など．
(5) 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．

(6) パック形式の 10 進数：PC や WS ではハード支援がなく性能が遅いが，メインフレームやオフコン中心に既存のファイル資産が多い．
(7) 副プログラムからのプログラム終了文．

10 おわりに

本論文では，分かち書きなしの日本語プログラム言語“まほろば”の文法を説明した．今後は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“初心者向け教育用言語”の仕様策定と処理系の実装．
(2) オブジェクト指向機能の日本語記述と，C++，Java，OOCOBOL を意識した標準的な仕様策定．
(3) C/C++，Java，COBOL の 3 つの既存環境への変換仕様の策定．これらの言語の生成機能を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ようになる．
(4) C/C++，Java，COBOL に対応する日本語プログラム言語仕様および処理系の作成．

謝辞 関連研究の参考資料の提供などご教授．ご協力いただいた宮脇富士夫教授（姫路工大），玉井哲雄教授（東大），大岩元教授（慶大），中鉢欣秀氏（慶大），伊藤康彦氏（九州日本電気ソフトウェア），佐藤孝夫氏（日電），増田和紀氏（キヤノンソフトウェア），床分眞一氏（日立）に感謝致します．1998 年秋と 1999 年冬のプログラミング・シンポジウムと 1999～2000 年のプログラミング研究会での報告時に大変有益な討論をいただいた方々に深く感謝致します．

参考文献

1) 今城哲二：＜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）．
2) 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)），p.168（1988）．
3) Tetsuo Tamai : On Japanese-based Programming , Journal of Information Processing, Vol.13 No.1, pp.49-56 (1990).
4) 今城哲二：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）．
5) 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）．
6) 今城哲二：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）．
7) 鈴木弘，中鉢欣秀，大岩元：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）．
8) 中鉢欣秀，大岩元：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）．
9) 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言語（日本語 C++）の開発，姫路工業大学工学部研究報告 No47，pp70-82（1994）．
10) 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）．
11) 加藤木和夫，畠山正行：日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）．
12) 兼宗進，久野靖：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・シンポジウム報告集，情報処理学会，pp.11-20（2001-1-9）．
13) 金子敬一，今城哲二，中村英夫：入門計算機ソフトウェア（第 5 章 言語処理系），朝倉書店，pp.76-129（2000）．
14) 西村恕彦，植村俊亮：入門 COBOL(新版)，p.236，オーム社（1993）．
15) 今城哲二：明解 COBOL，p.213，サイエンス社（1990）．

付録 まほろば記述例——まほろば0コンパイラの一部（字句解析）

注：かっこがネストしたとき、「…[1]「…[2]「…」[2]…」[1]…」のように内側のかっこにはネスト番号が、注釈にはイタリックで陰が印刷ツールで付与・変換されている。

［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」．
｛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］　番号：整数．
呼出し回数：常駐，列挙型，「最初，２回目以降」，初期値は最初．
｛文字コード表｝
文字コード種別表［６５９３６］：常駐，文字種別型，初期値は不当文字．
｛入力するソース関連の領域｝
入力行［１０００］：常駐，文字．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（）．
ファイルオープン（ソースファイル名）＝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←"　"．
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）＝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←"　"．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←入力行番号＋１．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］≠"　"のときは，ぬける．
入力文字位置←入力文字位置＋１．
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]
｛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］≠"　"のときは，ぬける．入力文字数←入力文字数－１．」[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＞入力文字位置のときは，入力文字位置←入力文字位置＋１．
それ以外は，入力文字位置←０．
もどる．
［終了］