コンピュータの言語

玉井 哲雄

コンピュータの言語というと,「2001 年宇宙の旅」に登場するコンピュータ HAL を連想する人が多いかもしれません.HAL は自然な英語で人間と話をします.映画の中で HAL の話す声は,あえて無機質で抑揚の少ないものにされていますが,会話は自然で,人の言葉をよく理解し,また自ら適切な文章を組み立てて話すという高度な言語能力を備えていることを示しています.しかし,2001 年が間近となった現在,このようなレベルの言語能力をコンピュータに与える技術は,まだとても完成の域に達していません.コンピュータに自然な言語能力を与えることは,人工知能という研究分野の大きなテーマで,活発な研究が行われています.しかしコンピュータの黎明期に楽観的に考えられたよりも,コンピュータに英語や日本語のような「自然言語を操る能力を与えることは,はるかに難しい問題であることが分かってきました.

しかし,コンピュータの言語として考えるべき課題は,いかに人間と自然に会話できるようにするかということだけではありません.実際,次のような重要なテーマがあるのです.

1. 人間がコンピュータに情報を伝達し,仕事をさせるためには,どのような言語を使うことが合理的か.
2. コンピュータが内部で思考(あるいはこのような擬人的な言葉を避けるなら情報処理)をする際に用いる言語は,どのようなものであるべきか.

そこで以下では,コンピュータにはやはり人間が使うのとは異なる何らかの固有の言語が必要であるという立場から,そのために創り出され,これからも創られていくであろう「人工言語に話の焦点を当てることにします.

コンピュータに意思を伝えるための言語

パーソナル・コンピュータを使うとしましょう.電源を入れます.ここから先は機種によりますが,たとえば現在最もよく使われている Windows というオペレーティング・システム (OS) が動いているものとすると,まずマウスを机の上で動かして,画面の左下隅のスタートという箱にカーソルと呼ばれる矢印を合わせ,そこでマウスのボタンをカチンと押すことから始めることになるでしょう.(最近のノートブック型パソコンでは,カーソルの移動にボールや小さなつまみのような別の形のものを使うものが多くなりましたが,動作は同じです.) そうすると画面にメニューが現れ,再びマウスを机上で移動することによりメニューの中からやりたいことを選び,マウスのボタンを押します.場合によってはそこでまた次の段階のメニューが現れ,そこから選ぶという動作を繰り返します.このような操作の後,使いたいプログラム,たとえば文書の編集や表計算やゲームのソフトウェアが起動されます.

あるいは「スタートの箱から始める代わりに,画面上に並んでいる図像 (アイコン) の中に適当なものがあれば,そこにカーソルを合わせてマウスのボタンを 2 度続けて押すことにより,直ちに目的のプログラムを起ち上げることもできるでしょう.

さて,ここで行った基本動作の組み合わせを,一種の言語と考えることができます.マウスでカーソルを移動する,ボタンを押す,ボタンを 2 度続けて押す,といった基本要素 (これを単語といってもいいでしょう) を組み合わせることにより,コンピュータに情報を伝え,指示をしているわけです.コンピュータはメニューを表示したりプログラムを起動することにより応答しています.

これだけでは，言語と呼ぶにはあまりにも単純かもしれません．しかしマウスのボタンにも 2 つ付いているものや３つ付いているものがあり，それだけでも言語の要素がずいぶん増えます．あるいはマウスのボタンを押したままマウスを動かすというちょっと複雑な操作もあります.（これらのやり方は「パソコンの説明書では，ポイント，プレス，クリック，ダブルクリック，ドラッグ，アイコンなどのカタカナ用語を多用して記述されますが，言葉としてもう少し神経を使えなかったものかと，残念な気がします．しかし，一度このような言葉が定着してしまうと，なかなか変えられません.）

さらに文字を入力して情報を伝えるとなると，いよいよ言語らしくなってきます．文字を入力する手段としては，古くは紙テープやパンチカード，新しくは音声 (人の声）や手書き文字などもありますが，なんといってもキーボードが長い間主役を続けています．日本語の文字入力にも，かな漢字変換が普及してからはキーボードが広く使われるようになりました．

文字を入力するからといって，自然言語を使うというわけではありません．よく日本語入力という言い方がされますが，正確には日本語文字入力であって，日本語でそのままコンピュータに意思を伝えるわけではないのです．ワープロのソフトウェアを使って日本語の文章を入力したとしても，入力された文章はコンピュータにとっては単に文字が並んだデータであり，その意味を理解することはありませんし，またその必要もないわけです．

それではキーボードでコンピュータに直接意思を伝えるのは，どのような場合でしょうか．Windows ではマウスによるやり取りが主体となりましたが，その前身の OS である MS-DOS では，キーボードで文字を入力することによりコンピュータに指示を出し，コンピュータも画面上に文字を表示して答えるというやり取りが主でした．Windows でも MS-DOS プロンプトというプログラムがあって，この従来からの方法によるやり取りが可能です．たとえば次のような命令をキーボードから出します．

```
copy file1 file2
```

file1 という名前のファイルの内容を，file2 という名前のファイルに複写します．

```
type textfile
```

textfile という名前の文字データの入ったファイルの中身を画面に書き出します．

```
format a:
```

a:で指定される装置（通常はフロッピーディスク）の中身を空にして，必要な初期設定を行います．

こうなると，だいぶ言語らしく見えます．実は，これらはコマンド言語と呼ばれる言語の文の例なのです．3 つの文はいずれも

動詞＋目的語１（＋目的語２）

という形式をとっています.「動詞」としては copy, type, format という英語の動詞が使われていますが，それは本質的ではありません．区別さえできれば c, t, f という１文字でもよかったかもしれませんし，○，×，△というような記号でも構わないわけです．漢字が許されるなら,「複写」「表示」「初期化といった言い方もありえるでしょう．

また，語順は英語の命令形と同じになっていますが，これも本質的ではありません．たとえば

目的語１（＋目的語２）＋動詞

という形でもよかったでしょう．この方が，日本人には馴染みやすいかもしれません．

いずれにせよ，この言語は人工的に定められた文法と語彙を持つものであるといえましょう．

プログラミング言語

上に挙げたコマンド言語の一つ一つの文は,「… をせよ」という簡素な命令 (コマンド) を表すだけの機能しかありません.コンピュータにより複雑な処理を行わせるためには,さらに豊かな表現力のある言語体系が必要です.そのような言語としてプログラミング言語があります.

プログラミング言語で書かれた記述を,プログラムといいます.プログラムはコンピュータによって解釈され実行されることにより,様々な動作を行います.ワープロも表計算もゲームも,みな何らかのプログラミング言語で書かれたプログラムです.また,コマンドの例としてあげた

```
copy file1 file2
```

という命令も,実はファイルを複写するという機能を持ったプログラムを起動するという意味を持つものです.type にも format にも,それぞれ対応するプログラムがあるわけです.

プログラミング言語には多くの種類がありますが,通常,条件によって場合分けをし,それぞれに応じて異なる動作をする条件文や,一まとまりの動作を繰り返し実行する反復文などの言語要素を備えています.また,取り扱う情報の構造を定義するためのデータ宣言文に相当する言語要素を持つのも普通です.

このような言語の構文や意味をどう定めたらよいかという問題は,まさに言語学の領域に入るものです.

チョムスキーの革命

米国の言語学者ノーム・チョムスキー (1928〜) は,1957 年に「文法の構造 (Syntactic Structures)」を発表し,そこで生成文法の理論を提唱して言語学の世界に革命をもたらしました.チョムスキーの理論は言語学にばかりでなく,認知科学やコンピュータ科学にも大きな影響を与えました.とくに,プログラミング言語の設計やその処理系の開発技術に対しては,理論的な基盤となっています.

そもそもチョムスキーの理論のきっかけの一つなったのが,コンピュータ科学の初期の代表的な理論的成果であるオートマトン理論です.オートマトンの一種で有限状態機械と呼ばれるものは,次々に入力される文字 (記号) を受け取り,それに従って状態を変えます.各状態で受理可能な文字が定められており,それらの文字ごとに次にどの状態に遷移するかという規則が定められているわけです.初期状態から出発して,入力された文字列を次々に受け取りながら,あらかじめ定められた最終状態 (の一つ) に至るとき,その文字列は受理されたといいます (図 1 参照).一つの有限状態機械を定めると,その機械が受理可能な文字列と受理不可能な文字列とが定まります.受理可能な文字列が一つの言語の文法的に正しい文を表し,受理不可能な文は文法的に誤った文を表していると見なすことができれば,有限状態機械が一つの言語を定義していることになります.このようにして定められる言語は,文字の連接,選択,繰り返し,という操作のみで構成される正規文法言語と言われる文字列集合と等価であることが知られています.

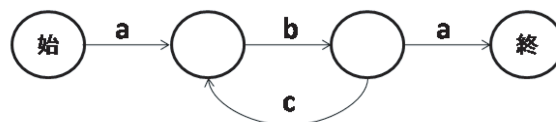

図 1: aba, abcba, abcbcba などの「文」を受理する有限状態機械

有限状態機械で定義できる言語は,かなり限定されたものです.たとえば,

```
ab, aabb, aaabbb, ...
```

のように文字 a が n 個続いた後，n 個の文字 b が現れるような文字列からなる言語を定義することができません．チョムスキーはより複雑な構造をもつ言語を形式的に定める方法として，句構造文法を与えています．句構造文法は，一連の書き換え規則として定められます．たとえば，次のような規則です．

```
(1) S -> NP+VP
(2) NP -> T+N
(3) VP -> V+NP
(4) T -> the
(5) N -> man, ball
(6) V -> hit, took
```

それぞれの規則は，矢印の左辺を右辺で置き換えることを意味します．大文字で表されている語は非終端記号と呼ばれ，Ｓから始めて非終端記号がなくなるまで，置き換えを続けます．そうすると，たとえば ”the man hit the ball” などの文が生成されます．

上に挙げた書き換え規則はすべて，左辺が非終端記号 1 つのみという特殊な形をしています．このような文法を文脈自由文法といいます．文脈自由文法で右辺の形にさらに制約を加えたものが，正規文法になっています．文脈自由文法では正規文法で書けなかった ab, aabb, ... のパターンの定義も書けますので（どうすればいいか，考えて見ください)，その言語はより広いものになっています．しかし，文脈自由文法でも表現できる範囲は限定されます．さらに左辺の形が非終端記号 1 つであるという制約を外したものが，文脈依存文法です．文脈依存文法によると，文脈自由文法では表現できない言語が定義できることが知られています．

現実の言語，たとえば英語の文法を表すには，文脈依存文法でもまだ不足です．たとえば，助動詞を含む表現の扱いや，能動態と受動態を等価に扱うことなどが，うまくできません．チョムスキーはそのための仕組みとして，変形文法というものを考案しました．生成文法と変形文法を組み合わせて，複雑な構造を持つ自然言語の文法を扱おうというのが，チョムスキー理論の骨子です．しかし，人工的な言語であるプログラミング言語の文法を定めるには，生成文法の範囲でほぼ十分です．

コンピュータの歴史の初期，すなわち 1950 年代に作られたプログラミング言語の Fortran(1957 年公表) や Cobol(1958 年公表) では，その言語仕様に生成文法は意識されませんでした．しかし，1960 年に発表された Algol から，その構文（syntax，言語学では統語論という用語を通常使います) を形式的に定義するのに生成文法の仕組みを意識的に用いるようになりました．

生成文法の理論は，プログラミング言語の文法を定めるだけでなく，その言語で記述されたプログラムを解釈しコンピュータで実行できる形に変換する処理系（コンパイラ）の作成技術にも大いに貢献しました．正規文法で正しい文を受理する有限状態機械が作られるように，文脈自由文法で正しい文を受理・解釈し，誤った文に対しては誤った個所を指摘するような機械 (プログラム) を構築する手法は，生成文法の理論を基盤として確立しています．

プログラム読本

これまではもっぱら構文 (統語論) の話でした．一般に言語では，統語論とともに意味論 (semantics) と運用論 (pragmatics) という 3 つの面があるといわれます．さらに通常の言語では音韻論 (phonology) ももちろん重要な側面ですが，プログラミング言語の場合は，少なくとも今までは，話し言葉として使われることはなかったのでその面は無視していいでしょう．

プログラミング言語の意味をどう定義するかはコンピュータ科学の重要なテーマであり，多くの理論的な成果が得られていますが，ここでは深く立ち入りません．通常の言語の場合，その意味を科学的に取り扱うことには本質的に困難な面がありますが，人工的な言語であるプログラミング言語の場合には，その意味を厳密に議論することは原理的に十分可能であるということだけを，指摘しておきます．

一方，言語の運用論とは，状況に応じて言葉がいかに使われるかという側面を表しています．話し言葉の場合はとくに，状況に応じて言葉の多義性がいかに解消されるかとか，敬語表現といった問題が議論が中心となりますが，書き言葉に類するプログラミング言語の場合は，文体，修辞学，言い回しというあたりが運用の範囲に入ってくるかもしれません．すなわち，文章の書き方についてさまざまな技法がありうるように，プログラムの書き方についてもよいスタイルや悪いスタイルが議論できるわけです．その意味で，プログラミングの本が文章読本に似た面を持ってくるのも自然なことです．人工的な言語であるプログラミング言語でも，その言語固有の言い回し，慣用句のようなものが生まれ，たとえばいかにも Lisp らしい書き方，Prolog らしい表現というものが存在するのは面白いことです（Lisp，Prolog はともにプログラミング言語の名前です）.

プログラミング言語で記述されるプログラムについて，自然言語で書かれた文章と同じように優劣を論じることに意味があるのは，プログラムの「読み手」がコンピュータだけでなく人間でもあるからです．つまり，プログラムはコンピュータが正しく解釈して実行しさえしてくれればよいというものではなく，人間が読んでも理解しやすく，その意味で美しく書かれなければならないのです．プログラムの読み手として第一に挙げなければならないのは，プログラムを書いた本人です．プログラムはそれを書く人が頭の中で考えたアルゴリズム（特定の機能を果たすための手順）をプログラミング言語で記述したものですから，ある意味で思想の表現といえます．そのためには誰よりもまず表現者自身が読んで理解できなければなりません.

さらに，他の人の書いたプログラムを読む必要が生じることも，日常茶飯事です．まず，多くのプログラムはチームによって共同開発されます．その際は，人のプログラムと自分のプログラムとの整合性をとるために，互いにプログラムを読み合う必要がでてくるでしょう．また，プログラムに問題点や誤りがないかどうかチェックするためにも，人に読んでもらうことはきわめて有効です．さらに，繰り返し使われるような標準的なプログラムの単位がライブラリとして用意されており，そこから必要なものを選んで使うということもよく行われますが，その際もライブラリにあるプログラムの中身を読んで，自分の意図に合うものかどうかを確かめることが必要になるかもしれません．別のケースとしては，現在使われているプログラムの機能を拡張したり変更したりする作業が必要となることがあります．この場合も，他人のプログラム（あるいは自分のプログラムかもしれませんが，しかし過去に書いて記憶の定かでないもの) を読んで，どこにどう手を加えたらよいか，判断する必要があります．このような仕事を保守作業と呼びます.

さらに，よいプログラムを読むことは，プログラミング能力を向上させる大変よい手段であるといえます．この点は，数多くの「文章読本で言っていることが，結局よい文章を読めということに尽きているのと，事情が似ているでしょう．優れたプログラムは作品としての鑑賞に堪えるものです．もちろん，普通の文章と違ってコンピュータ上で動作することに意味があるので，美の基準は同一ではないでしょう．読み手には，表現を鑑賞するだけでなく，その動作を考えながら論理を追うという努力が要求されます．しかし，よいプログラムを書く能力は，かなりの程度よい文章を書く能力と繋がってくるようです．その名も The Art of Programming という大著を 30 年前から書き進めているドナルド・クヌースが優れた文章家であるのも，偶然ではないでしょう.

プログラミング言語による記述例

最後に少しだけ，代表的なプログラミング言語で書かれたプログラムの例を挙げておきましょう．ある程度の長さのあるプログラムを載せるだけのスペースがないので，とても鑑賞に堪えるような作品という訳にはいきません．簡単なプログラム例として，与えられた年が平年か閏年かを判断するという問題を取り上げましょう.

現行のグレゴリオ暦 (1582 年に制定．日本では 1872 年に採用) では，年数が 100 の倍数の時は 400 で割り切れる年，100 の倍数でない時は 4 で割り切れる年を閏年，ほかを平年とします (「岩波 理化学辞典」より)．つまり，1996 年や 2000 年は閏年ですが，1997 年や 2100 年は平年です．いきなりプログラムを見せましょう．まず，大学のプログラミング教育でよく用いられる言語である Pascal で書いた例です.

```
program LeapYear(input,output);
var year : integer;
begin
  write('Which year? '); readln(year);
  if year mod 100 = 0 then
    if year mod 400 = 0 then writeln('Leap year.')
                        else writeln('Common year.')
  else
    if year mod 4 = 0 then writeln('Leap year.')
                      else writeln('Common year')
end.
```

詳しい説明は省きますが，このプログラムは西暦の年が入力 (readln という文を使っています) されると，上に述べた判定法に素直にしたがって結果を出力 (writeln という文を使っています) しています．同じ Pascal でも，閏年かどうかを判定する部分を独立した関数という単位で書いてみると，次のようになります．なお，判定法をまとめて少しすっきりさせました．

```
program LeapYear2(input, output);
var year : integer;
function isLeap(y : integer) : boolean;
  begin
    if ((y mod 4 = 0) and (y mod 100 <> 0)) or (y mod 400 = 0)
      then isLeap := true
      else isLeap := false
  end;
begin
  write('Which year? '); readln(year);
  if isLeap(year) then writeln('Leap year.')
                  else writeln('Common year.')
end.
```

次は，インターネット上で使われるプログラムを記述するのに適したプログラミング言語として最近はやりの Java で書いた例です．

```
class LeapYear {
  public static void main(String args[]){
    int year = Integer.parseInt(args[0]);
    if (isLeap(year)) System.out.println("Leap year.");
    else System.out.println("Common year.");
  }
  static boolean isLeap(int y) {
    if ((y%4 == 0) && (y%100 != 0) || (y%400 == 0)) return true;
    else return false;
  }
}
```

Pascal（の前身の Algol）にしても Java にしても，さらに Lisp，Fortran，Cobol，C などのほとんどのプログラミング言語も，すべて英語圏で作られました．そのため if とか return といったキーワードを使いま

すし，文法的にも英語の影響を多かれ少なかれ受けています．たとえば日本語をベースとしたプログラミング言語はないのでしょうか．実はあります．次に示すプログラムは，Mindという日本語ベースのプログラミング言語で書いたものです (言語の名前は，なぜか英語風です) .

```
閏年?とは
    注目年は　変数
    注目年に　入れる
    注目年の　4での　余りが　ゼロ?
      かつ
    注目年の　100での　余りが　ゼロ以外
      または
    注目年の　400での　余りが　ゼロ?。

閏年判定とは
    指定年は　変数
    指定年入力をして　指定年に　入れる
    指定年が　閏年?
        ならば　「閏年」と　表示し
        さもなければ　「平年」と　表示する。
```

以上で挙げたプログラムはあまりに簡単で，とても文章読本の見本になるようなものではありません．これをきっかけにして，より進んだプログラミングの本を読み，プログラミングの醍醐味を味わう人が一人でも増えることを願っています.

執筆者紹介

玉井哲雄（たまいてつお）
1948 年 神奈川県逗子市生まれ
［専門］ ソフトウェア工学，知識工学
［著書］『ソフトウェアのテスト技法』（共著，共立出版）,『ソフトウェア/アルゴリズムの権利保護』（共著，朝倉書店）,『線形計画法の実際』（共著，産業図書）など.
［一言］ 駒場に来たのが 1994 年の 4 月で，まさに『知の技法』が出版され大きな話題となった時である．正直なところ今度書くことを引き受けるまでこれを読んでいなかったが，この機会に「知の三部作」を一気に通読して大いに楽しんだ．今回は書く方で楽しんだが，読者に楽しんでもらえるかどうかは予測がつかない.

「知の未来ここに！」

本文でも述べたように，ソフトウェアはある種の言語による記述として作成されます．その意味でソフトウェアは，抽象性を持った著作物という性格を持ちます．一方で，ソフトウェアはコンピュータを通じて動作し，世界に対して直接的な働きかけを行います．その意味では，工業製品や人工システムと見なすことができます．このような抽象性と実際性を合わせ持つものは，人類がものを作ってきた歴史の中で他にあまり類例がないものといえましょう．優れたソフトウェアを作ることは，知にとっての大いなるチャレンジなのです．しかし，世上でよく言われるように，日本発のソフトウェアで世界的に普及しているものが，きわめて少ないことは事実です．インターネットに象徴されるように，コンピュータの世界では国の違いというのは大きな意味を持ちません．しかしそれだからこそ，優れたソフトウェアはどこで作られたものであろ

うと世界で評価される可能性が高いのです．私自身はソフトウェアの方法論を研究の対象としていますが，ソフトウェアという知にとって総合的な技術の要求される分野に，多くの若い人が挑戦してほしいと思っています．