日本語プログラミング言語の品格

岡田　健
慶應義塾大学　政策・メディア研究科

品格ある日本語プログラミング言語の文法について考察する．一般的に日本語プログラミング言語は，プログラミング初学者には好意的に受け止められ，熟練者には気持ち悪くて品格の無いものとして捉えられる．この受け止め方の違いは，プログラムを読解するときの字句解析・構文解析のやり方，そして語順が影響している．初学者と熟練者の双方に好意的に受け止められる，品格ある日本語プログラミング言語を開発するために，少ない文法ルールで表現する文法の開発，スタック型言語の開発を模索する．

1. はじめに

日本語プログラミング言語は古くから研究され，いくつかの実用的な言語も開発・利用されている．著者も日本語プログラミング言語を開発して[1) 2) 3)]，教育現場で活用して学生などのプログラミングの初学者から「これなら私でも理解できそう」「面白そう」と好感触を得ている．

日本語プログラミング言語の存在意義を否定する意見も古くから聞かれる．熟練したプログラマーであればあるほど日本語プログラミング言語に対して感覚的な否定意見が見られることが多い．時には日本語のソースコードに対して「気持ち悪い」という感想が出てくることもある．

初学者と熟練者の日本語プログラミング言語に対する意見の相違の原因のひとつは，ソースコードの解釈方法の違いにあるのではないだろうか．ソースコードを解釈する際には，初学者も熟練者も，字句解析と構文解析を通して構造を読み取る．その解釈方法の違いによって評価の極端な違いが生まれると仮定して，本論文では初学者と熟練者による日本語プログラミング言語のソースコード解釈の違いについて論じる．

2. 初学者によるソースコード解釈

例として同じ目的を持つ，２つの言語で記述されたプログラムを挙げる．どちらも目的は入力された数値が奇数か偶数かを判定するプログラムであり，図 1 は Java 言語で書かれ，図 2 は日本語プログラミング言語「言霊」で書かれている．

```
1: int input = keyInput();
2: if( input%2 == 0 ){
3:   System.out.println("偶数です。");
4: }else{
5:   System.out.println("奇数です。");
6: }
```

図 1　奇数偶数判定プログラム(Java)

```
1: キー入力して、それを input とする。
2: input％２が０ならば｛
3:　「偶数です。」を出力する。
4: ｝をして、そうでなければ｛
5:　「奇数です。」を出力する。
6: ｝をする。
```

図 2　奇数偶数判定プログラム(言霊)

初学者はプログラミング言語の概念や構文の知識がなければ，ソースコードを読んでも理解できないはずである．そういう意味で図 1，図 2 のどちらを読んでも理解度に大差は無い．

初学者は図 1 のソースコードを見せると「難しそう」「私には無理そうだ」と自分の理解能力に絶望し，時には激しい拒否反応を示す．一方で初学者は図 2 のソースコードには「何となく理解できそう」と楽観的な反応を示すことが多い．

初学者のこの反応の違いは，未知の言語を前にして，既存の知識を使って解釈を試みていると考えられる．既存の知識による言語解釈には，以下の要素が含まれる．

- 字句解析による解釈
- 構文解析による解釈
- 語順による解釈

以下の節では，初学者による図 1 と図 2 の反応の違いが起こる理由を，この 3 要素による言語解釈によって論じる．

2.1.　字句解析による解釈

どんな自然言語・人工言語に関わらず，言語を解釈する際には文を字句に分解する．初学者が未知の言語を目の前にした時，この字句解析の成功の度合いによって，言語を理解できそうと感じたり感じなかったりする．

```
int if else = ( ) { } ; % == . ″
```

図 3　図 1 の解釈に必要な予約語

図 1 のプログラムを字句に分解する場合，図 3 が予約語であり，それ以外の字句はユーザが定義したシンボルであることを知っている必要がある．だが初学者にはその知識が無く，図 1 のプログラムを目の前にした時にまずどこで区切ったらよいのか分からない．熟練者がこの時の初学者の気持ちを想像したかったら，アラビア語で記述された文章を読むと良い．我々は一般的にアラビア語の単語区切りは全く分からない．

```
％ ｛ ｝ 、 。 を と が ならば ならば そうでなければ
```

図 4　図 2 の解釈に必要な予約語

図 2 のプログラムを字句に分解する場合，図 4 が予約語であり，それ以外の字句はユーザが定義したシンボルであることは，自然と理解できる．字句文法が自然言語である日本語と共通しているため，予備知識が無くてもプログラムを字句に分割出来る．図 4 の中には初学者では理解できないであろう記号（％{}）が含まれているため完全に意味が類推出来るわけではないが，プログラムの他の部分の多くが字句解析出来るため，比較的理解に対して楽観的な印象を持ちやすい．

2.2.　構文解析による解釈

言語を解釈する際には構文解析によって文の構造を理解する．初学者が図 1 と図 2 の２つのプログラムを前にしたときも，構文解析が出来るか否かによってプログラム言語に対する印象が変化する．

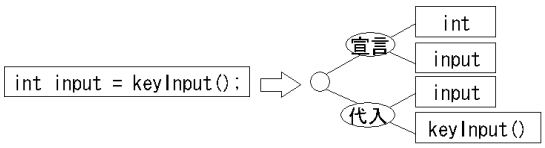


図 5　図 1 のプログラム（一部）の構造

図 5 は，Java で書かれた図 1 のプログラムの一部の構造を図式化したものである．プログラムには宣言文と代入文の２文が含まれ，各文には要素が２つあるという構造をしている．だがこうした構造は初学者には予想が出来ない．

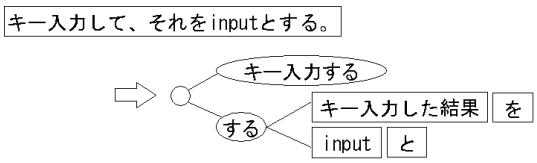


図 6　図 2 のプログラム（一部）の構造

図 6 は，言霊で書かれた図 2 のプログラムの一部の構造を図式化したものである．初学者でもソースコードを読むだけで，文の数，１文の範囲と含まれる要素を理解して，文構造を作ることが出来る．助詞により文における名詞と動詞の関係性が分かり，句読点や送り仮名によって文の区切りと関係性が理解できるためである．

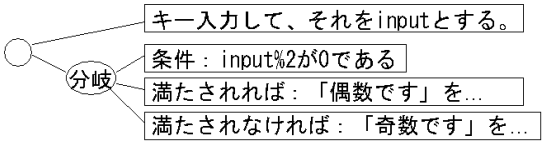


図 7　図 2 のプログラム（一部）の構造

図 2 は分岐構造やブロックなど初学者には未知の構文要素を含んでいるが，「～ならば～をする」という自然言語である日本語の知識からの類推により，図 7 のような構造を見出すことは可能である．

なお図 7 には%演算子という初学者に理解できない記号があるが，それにより構造の理解が出来なくなるとは考えにくい．多少分からない記号がありつつも，全体としては日本語の文法に則って記述されているため，文構造は理解できると考えられる．

2.3.　語順による解釈

Java をはじめとする一般的なプログラミング言語はヨーロッパ言語を使う人間が設計していて，印欧語の影響を強く受けている．そのため図 8 のような「どうする，何を」という印欧語の語順で記述される．

```
printf( "Hello World!" );
```

図 8　HelloWorld プログラム(C 言語)

しかし日本語は「何を，どうする」という語順であり，印欧語とは異なる語順である．そのため日本人が図 8 のプログラムを解釈する際には，語順の変換を行う必要がある．

印欧語の語順から日本語の語順への変換はコストがかかることが予想される．図 9 を例に挙げる．印欧語の語順で記述された数式を読み，我々は「２に３を掛けて，４を足す」などと解釈する．これは中値記法から後置記法への変換であり，この変換を行うのにスタックが一つ必要である．

```
2 * 3 + 4
↓
2 3 * 4 +
```

図 9　中値式から後置式への変換

日本語の語順である後置式に変換されてから，さらにそれを解釈する際にもスタックが必要になる.図 10 はその模様である．

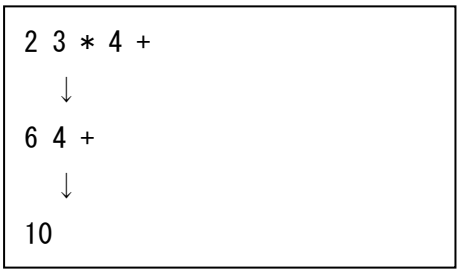


図 10　後置式の演算

このように日本人が「どうする，何を」の語順の式を解釈しようとすると，頭の中で二つのスタックを用意して演算を行っていることが予想される．日本語を解釈するのと比べると二倍のスタックを頭の中に用意する必要があり，解釈する際の障壁となる．

日本語で書かれたプログラムであればスムーズに解釈が出来る．そもそも語順の変換が必要ないため，解釈に必要なスタックも一つで済む．個人的な体験として，日本語でプログラムを書いているとある種の気持ちよさを感じることがある．既存のプログラミング言語と比べて少ないスタックで思考できるため，脳に対する負担が少ないことが予想される．

3.　熟練者によるソースコード解釈

熟練者は Java で書かれた図 1 と言霊で書かれた図 2 のプログラムを見たとき，初学者とは異なる解釈をする．その解釈の違いによって初学者とは反対の反応を示す．

3.1.　字句解析による解釈

熟練者はプログラミング言語の字句解析に慣れているため，言霊で書かれた図 2 のプログラムも明確なルールを見出して字句解析を試みる．ところが日本語プログラミング言語を見ただけで明確な字句解析ルールを見出すことは難しい．日本語を母語としている人にとって，字句の分割方法は複数思い付いてしまう．

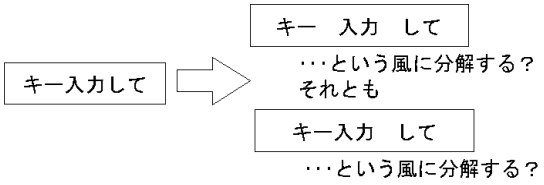


図 11　字句解析に戸惑う例

例えば図 11 のような日本語表現を見たとき，熟練者はどのように字句を分割したらいいか戸惑うことになる．日本人なのでプログラムの意味は理解できるのだが，プログラミング言語として見たときに字句解析が出来ないため，理解できそうでよく分からないという曖昧な印象を持ってしまう．

3.2.　構文解析による解釈

熟練者が日本語プログラミング言語を構文解析する際にも，どのように構造を解釈すべきか戸惑う例がある．

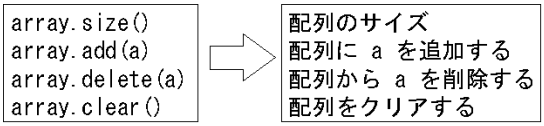


図 12　レシーバに付く助詞が変化する例

図 12 は日本語で記述されたプログラムを読むと，元のプログラムの構造が理解しにくくなる例である．左側はどれも array がレシーバであることが記号「.」であることが明確である．だが右側の日本語は，配列がレシーバであることが分かりにくい．語句「配列」に付属する助詞が「の」「に」「から」「を」と変化している．そのため助詞からレシーバを判断することが出来ないため，手続き呼び出し文の構造を判断することが難しくなってしまう．

3.3.　語順による解釈

2.3 では印欧語の語順「どうする，何を」を日本語の語順「何を，どうする」に変換

して解釈するためにスタックを余計に消費することを述べた．だが熟練者は印欧語の語順で記述されたプログラミング言語に慣れていて，意味を解釈する際にわざわざ日本語の語順に変換をしていないのかもしれない．そのため日本語の語順で書かれた日本語プログラミング言語を見ると，熟練者は日本語の語順から印欧語の語順に変換するという逆転現象が起こっている可能性もある．

4. 品格ある日本語プログラミング言語を作るために

日本語プログラミング言語は初学者と熟練者で認識に著しい違いがあり，初学者には肯定的に受け止められるが熟練者には「品がない」「気持ち悪い言語だ」と受け止められる．今後日本語プログラミング言語を作るとしたら，初学者と熟練者の両方から品格ある言語であると認められなければならない．

本章では新しい日本語プログラミング言語を作るための方針を２つ述べる．

4.1. シンタックスを限定する

熟練者が日本語プログラミング言語を読んだときに，字句文法，構文が曖昧に見えてしまうために字句解析・構文解析ができないことを 3.1, 3.2 で述べた．そこでシンタックスを限定し，以下の文法に従ってプログラムを日本語で記述する試みを行った．

この文法は未だに研究中であるが，これにより生まれるプログラムは初学者でも読み下せて，熟練者でも少ないルールを理解すればプログラミング言語として読解することも可能である．

```
<プログラム>::=<文>*
<文>::=
   <値>を<動詞>する。 |
   <値>に<動詞>する。 |
   <値>を<値>と<動詞>する。 |
   <値>を<値>に<動詞>する。
```

図 13　シンプルな日本語プログラミング言語の文法

ただし文法ルールが少ないため，日本語として長ったらしくて冗長な表現になるという弱点がある．文法ルールを以下に少なくしつつ，ストレスを感じない程度にコンパクトな表現にするのが今後の課題である．

```
1 １を繰り返し回数と名付ける。
2 １をfibAと名付ける。
3 ２をfibBと名付ける。
4 fibA+fibBをfibCと名付ける。
5 [
6     java.lang.Systemのoutに対して、fibAを渡して、printlnを命令する。
7     fibAをfibBに変える。
8     fibBをfibCに変える。
9     fibCをfibA+fibBに変える。
10    繰り返し回数を繰り返し回数＋１に変える。
11    繰り返し回数＝２０ならば [
12        脱出する。
13    ]
14 ] を繰り返す。
```

図 14　図 13の文法を使って記述したプログラム例

4.2. スタック型言語として作る

熟練者が日本語プログラミング言語を読んだときに戸惑うのは，既存のプログラミング言語を日本語の糖衣構文で隠されているあまり，陰に隠されているセマンティクスが分かりにくいためだと考えることが出来る．

そこで日本語を糖衣構文として活用するのではなく，自然言語としての日本語をそのままプログラミング言語として活用するため，スタック型言語として実装する試みを行った．

日本語プログラミング言語をスタック型言語として実装した例として Mind[4)5)] が過

去にある．Mind では助詞や接続詞にセマンティクスにおける意味は与えなかったが，今回の試みでは全ての品詞をスタック命令として解釈する．例えば図 15 のように，助詞はスタックに積まれた情報に対してタグ付けするスタック命令として実装できる．

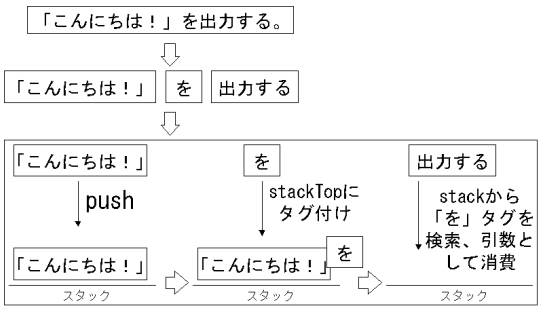


図 15　スタック型言語として実装した日本語プログラミング言語の解釈例

5. シンポジウム当日の質疑応答

質問：(一橋大学　長慎也)

語順の問題がクリティカルなのは何となく理解できるが，アメリカ人が語順の問題で悩んでいるような実例はあるか？

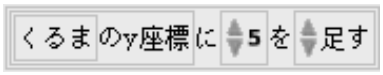
解答：

そういったデータは見たことはないが，私が「ことだま onSqueak」[6]を開発したときに面白いことがあった．Squeak は英語の語順で命令タイルが作成されていて，日本語版 Squeak はその語順のまま語句だけを日本語に変えていた．これに対して「ことだま on Squeak」は語順を日本語に変えた．

Squeak：

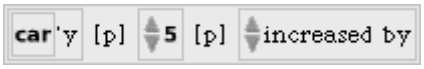
日本語版 Squeak：

ことだま on Squeak：



「ことだま on Squeak」の開発したことを報告するためアラン・ケイ博士を訪ねたときに，語順の違いがクリティカルであることを伝えるために，日本語の語順で掻かれた命令タイルを，語句だけ英語にしたものを提示した．日本人が日本語版 Squeak で使っているものを，アメリカ人の立場で理解できるようにした．なお，以下の表記における[p]は particle（助詞）を表している．

ことだま on Squeak 英語版：



その結果，とても気持ち悪いという反応をしていた．語順が思考に関係しているのだと考えている．

質問：(千葉商科大学 久保裕也)

プログラミング言語の教育効果を測定するために，前後半などで使用するプログラミング言語を分けてみたら？　様々なタイプごとに分けて，結果を評価してみたら？

解答：

教育効果を計るためにグループ分けをするのはよいが，そのために一部の学習者には教育効果が低いと予想される言語を使用させる必要がある．その事を学習者はどう考えるだろうか．なかなか難しいのではないだろうか．

（一橋大学　長慎也）プログラミング言語を教える順序を変えることによって実験することも出来る．

またプログラミングを学ぶためなら，プ

ログラミング言語は何でも良いのでは，と考えて，教えてみたこともある．

質問 : (株式会社インターネットイニシアティブ　和田英一)

戦争中は敵性言語が使えなくて，ＡＢＣを使う代わりにいろはで全てを記述していたが，結局そういった表現が普及することは無かった．

プログラミング言語も英語で記述されているわけではなく，アルファベットを使用しているだけ．欧米の人も身構えて使っているのではないだろうか．日本人も記号を使うときにアルファベットが使えると，「これから記号を使うぞ」という態度になる．

解答 :

記号を使うときにアルファベットを使えた方が良いのは同意する．私自身も日本語プログラミングで記述するときも，変数名にアルファベットは頻繁に使う．

質問 : (株式会社インターネットイニシアティブ　和田英一)

言葉に違いがあるのに，意味に違いが無いのはとても気持ち悪い．例えばセミコロンがあってもなくてもいいとか，「〜する」「〜し」「〜して」の差が無いのは気持ち悪い. 曖昧で通じるが, 曖昧だと気持ち悪い.

解答 :

語尾が「〜する」「〜し」「〜して」のいずれの表現も解釈する日本語プログラミング言語を作ったことがある．そしてそのような表現が最適だと考えているわけではない．日本語プログラミング言語を洗練させるため，曖昧さを生む不要な表現を削ぎ落としていく作業が必要だと考えている．

質問 : (株式会社インターネットイニシアティブ　和田英一)

それなら最初からそれを作ればいいのではないだろうか.

解答 :

日本語の思考方法を大切にしたいと考えているため，まずは自然言語としての日本語に近いプログラミング言語を作っている．例えば語順は，日本語らしい思考方法としてとてもクリティカルな問題である．

質問 : (株式会社インターネットイニシアティブ　和田英一)

語順も変わってもいい，という点が気持ち悪い．「椅子に腰を掛ける」「腰を椅子に掛ける」のいずれも解釈出来るというのは気持ち悪いのではないか.

解答 :

動詞の「掛ける」が末尾に来る点は変わらない．そこが日本語の語順として重要なところだ．また名前引きの引数という考えが既存のプログラミング言語にもある．その例では「椅子に」の助詞「に」が名前引きされると考えれば，曖昧ではなくなるのではないだろうか.

6. おわりに

一般的に日本語プログラミング言語は，プログラミング初学者には好意的に受け止められ，熟練者には否定的で気持ち悪い，品格の無いものとして受け止められがちである．その受け止め方の違いは，ソースコードを読解する際の字句解析，構文解析，語順の差によって生じる．

初学者は Java などの既存言語では全く

字句解析・構文解析が出来ないが，日本語プログラミング言語の場合は自然言語としての日本語の知識を使って，字句解析と構文解析が可能である．また語順も日本語と同じであるため，解釈に必要なコストが低くなる．

熟練者は日本語プログラミング言語を見ると，文法ルールがすぐに理解できないためにどのように字句解析・構文解析をしたらよいか分からず，気持ち悪いと感じてしまう．

新しく日本語プログラミング言語を作るために，２つの方針を立てて研究を進めている．１つは少ない文法ルールで全て表現することを試み，もう１つはスタック型言語として全ての文法要素をスタック命令として実装することを試みている．

謝辞：プログラムシンポジウム当日に有意義な質問をして下さった方々に感謝いたします．

参考文献

1) 岡田健，中鉢欣秀，鈴木弘，大岩元：「日本語プログラム言語「言霊」」，情報処理学会 2002FIT(2002)

2) 岡田健，大岩元：「プログラミング言語としての日本語 慶應義塾大学湘南藤沢学会」 Keio SFC Journal, Vol.2 No.1 pp114-134 (2002)

3) 岡田健，大岩元：「日本語プログラム言語「言霊」におけるメソッドの記述方法」情報処理学会第４６回プログラム・シンポジウム(2005)

4) 木村明，片桐明：「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．プログラミング言語」 16-4、情報処理学会 pp.25-32(1988)

5) 西之園晴夫：「情報教育用言語としてのMind の文法と記述法」日本科学教育学会年会論文集 Vol.13(19890805), pp371-374 (1989)

6) 杉浦学，松澤芳昭，岡田健，大岩元：「アルゴリズム構築能力育成の導入教育：実作業による概念理解に基づくアルゴリズム構築体験とその効果」:情報処理学会論文誌 Vol.49 No.10 pp.3409-3427(2008)