プログラミングの初歩概念を学ぶための日本語プログラミング環境

岡田 健[†] 杉浦 学[†]
松澤芳昭[†] 大岩 元[††]

教育用プログラミング環境「日本語 Squeak」を改良して、初心者がプログラミングの様々な概念を学ぶのに適したプログラミング環境「ことだま on Squeak」を開発した。「ことだま on Squeak」は自然言語としての日本語が持つ表現を利用してプログラムを表記しており、学習者はプログラムを読解することで、プログラミングにおける順次構造・分岐構造や代入の概念の理解が進むように工夫されている。

1. はじめに

情報処理学会による「日本の情報教育・情報処理教育に関する提言2005」[1]（以下、提言と呼ぶ）が発表され、情報教育と共に情報処理教育の重要性を指摘している。情報処理教育では「手順的な自動処理」の構築が重要視され、実際の教育ではプログラミング教育が行われることが予想される（提言では必ずしも教育の手段としてプログラミングのみに限定しているわけではないが、一般的にはプログラミングを行うことが多いと考える）。

情報処理教育で中心となるのは、問題を定義して解決のためのアルゴリズムを自ら考え、そして実践する能力の育成である。問題解決のためにプログラミングを行うのであって、プログラミング言語の文法を学ぶことや些末なテクニックを学ぶことが目的ではない。

我々はこうした情報処理教育で使用されるプログラミング言語は、日本語プログラミング言語を用いて行うのが理想的であると考えている。問題を考え、アルゴリズムを考える際には母国語である日本語を用いる。アルゴリズムからプログラミング言語に変換する際に、CやJavaを用いると翻訳のコストが非常に高くなってしまう[2),3)]。

我々は日本語Squeak[4]を改造して、日本語プログラミング環境「ことだま on Squeak」を開発した。「ことだま on Squeak」は日本語としてコードが読めて、受講者がアルゴリズムとして考えたものをそのままコードとして実現できる環境を目指して開発された。

本稿では「ことだま on Squeak」における様々な、プログラムの概念を日本語として記述する手法について提案する。

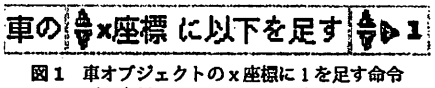

図1 車オブジェクトのx座標に1を足す命令
（日本語 Squeak における表記）

図2 車オブジェクトのx座標に1を足す命令
（「ことだま on Squeak」における表記）

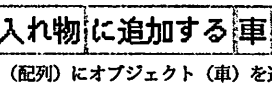

図3 入れ物（配列）にオブジェクト（車）を追加する命令
（日本語 Squeak における表記）

2. 「ことだま on Squeak」における日本語表記

本章では「ことだま on Squeak」において、どのような工夫により日本語として読めるようにしたのか、その工夫を紹介する。

2.1 日本語として読解するための記述方法

日本語として違和感なくコードを読解するために必要な要素は、助詞・助数詞と、日本語語順を使って記述することである。説明のために日本語 Squeak の表記と「ことだま on Squeak」の表記を比較して議論する。

2.1.1 語順の変更

日本語の語順を用いることで、円滑にコードを理解させることが出来る。

日本語の語順は、目的語が文の前半に来て動詞が最後に来るのが一般的である。この一般的な文法規則に反する図1は、一読して理解することが難しい。特に文末に来ている「1」の意味を理解するのは困難である。図2のように、動詞を末尾に配置するだけで「1」が表す意味が理解しやすくなる。

2.1.2 助詞の使用

助詞を使用することで、コード中に登場する概念の役割を強調することが可能になる。

† 慶應義塾大学政策・メディア研究科
Keio University, Graduate School of Media and Governance
†† 慶應義塾大学環境情報学部
Keio University, Faculty of Environmental Information

図4 入れ物（配列）にオブジェクト（車）を追加する命令
（「ことだま on Squeak」における表記）

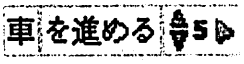

図5 オブジェクト（車）を前進させる命令
（日本語 Squeak における表記）

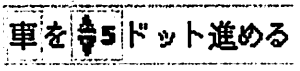

図6 オブジェクト（車）を前進させる命令
（「ことだま on Squeak」における表記）

例えば図3と図4は、どちらも配列にオブジェクトを追加する命令である。図3の表記では、「車」という言葉に対して助詞が付いていないために、命令の中で「車」がどのような位置づけなのかが読み取りにくい。「ことだま on Squeak」（図4）では「車」に対しても助詞「を」を付けることで、車が配列に追加しようとしている対象を指していることを明確にしている。

2.1.3 助数詞の使用

助数詞を使用することで、コード中に現れる概念の型を強調することが可能になる。

例えば図5と図6は、どちらも車を数ドット前進させる命令である。図5の表記では、末尾に書いてある「5」が一体何を指しているのかが理解できない。「ことだま on Squeak」（図6）では助数詞として「ドット」を付けることで、「5」が距離を表していることを示している。

「ことだま on Squeak」を使って授業を行うと、生徒は「5」が何を表しているのかを考えようとした実例がある。ある生徒は図6を見て教師に対して「ドットって何ですか?」という質問を投げかけてきた。これは生徒が日本語の表現を見て「5」が一体何を表しているのかを考えようとしている証拠である。教育現場で使用されている多くのプログラミング言語の場合は、このように表記の一部を取り出して「これはどういう意味ですか?」と考えることは起こりにくい。意味の分からない表記が多すぎて、質問のしようがないためである。

3. 順次構造を理解するための記述方法

「ことだま on Squeak」は、順次構造を正しく理解させるために自然言語としての日本語表現を活用している。

「ことだま on Squeak」は順次構造を表すために、命令文の末尾を連用形として表現している。図7は、「ことだま on Squeak」において命令タイルを組み合わせたときの日本語表記の変化を表している。図7

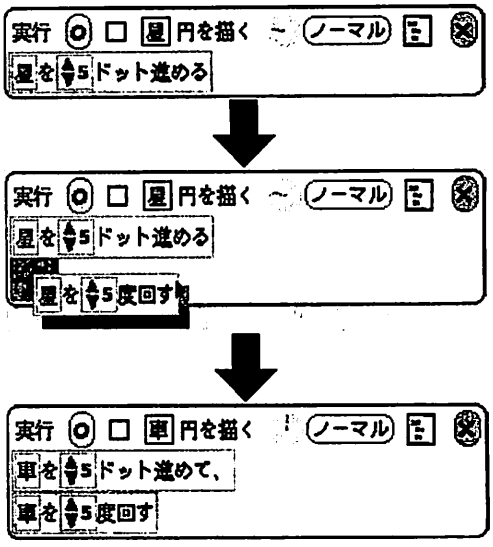

図7 命令タイルを追加した時の、表記の変化
（「ことだま on Squeak」における表記）

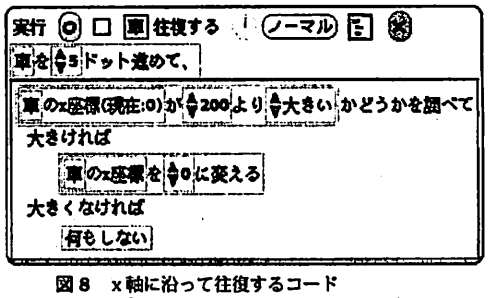

図8 x軸に沿って往復するコード
（「ことだま on Squeak」における表記）

の上図ではひとつしか命令がないため、動詞は「進める」と終止形で表現されている。だが図7中図のように命令タイルを追加すると、図7下図のようにひとつめの命令タイルの動詞が「進めて、」と連用形に変化する。

自然言語としての日本語が持っている性質のひとつである動詞変化を採用することで、「ことだま on Squeak」は順次構造を理解しやすいプログラミング環境を提供している。「～をして、～をする」と連用形を使って命令と命令をつなげると、自然と「ひとつめの命令を実行してから、ふたつめの命令を実行する」という実行順序を意識することになる。

4. 分岐構造を理解するための記述方法

「ことだま on Squeak」は分岐構造を正しく理解させるために、自然言語としての日本語表現を採用している。

分岐構造を正しく理解するために重要なことは、条件節で何らかの処理が行われていることを理解させることである。図8は、条件節で処理が行われてから分岐が発生することを説明するための日本語表

```
if( keyPressed() ){
  fire();
}
```

図 9　Java における条件分岐の表記

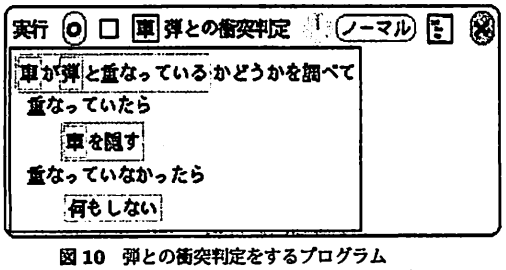

図 10　弾との衝突判定をするプログラム
（「ことだま on Squeak」における表記）

記を表している。

図 8 のプログラムは、まず最初に「車を 5 ドット進めて、」それから分岐が行われる。分岐ではまず「車の x 座標が２００より大きいかどうかを調べ」る。これを調べた後に、大きいか大きくないかで次に実行する処理を選んでいる。以上のような条件分岐の解説に匹敵する説明が、プログラムの中にそのまま表現として使用されている。

分岐構造を表現しているために重要なことは 2 点ある。ひとつめは、「～かどうかを調べて、」という表現を採用することにより、条件分岐の前に処理が存在していることを明示していることである。ふたつめは、条件成立の条件が具体的な「大きければ」「大きくなければ」と表現されていることである。

初心者が陥りやすい間違いのひとつは、条件節の中の記述は静的な条件であると勘違いしてしまうことである。筆者は Java を使って分岐を解説しているときに図 9 を生徒に示して、「日本語で処理を説明してみなさい」と指示したところ、生徒は「keyPressed を実行しているかどうかを調べて、もしも keyPressed が実行されていたら fire する」と答えた。条件節に書いてあることを静的な条件であると勘違いしてしまい、keyPressed というメソッドが実行されることに気付かない。

条件節に記述された条件は、それが真であるかどうかを調べるという処理が最初に行われる。図 9 に関する質問で、私が欲しかった回答は「keyPressed を実行して、その戻り値が true であるかどうかを調べて、true ならば fire する」という実行順序である。条件節を表現するには「～を調べて」という表現が先頭に来るのが適切であると考える。

分岐構造を表現するために重要なことのふたつめは、条件成立の条件が具体的な表現として記述されていることである。図 10 は弾と衝突しているかを判定するプログラムであるが、成立時の条件「重なっていたら」と不成立時の条件「重なっていなかったら」が記述されている。このように成立条件を具体的に記述することで、初心者でも分岐構造を理解しやすくすることが可能である。

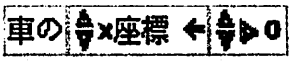

図 11　代入文を表す命令タイル
（日本語 Squeak における表記）

図 12　代入文を表す命令タイル
（「ことだま on Squeak」における表記）

5. 代入文を理解するための記述方法

「ことだま on Squeak」は代入の概念の理解を助けるために、代入文を日本語で表現している。

一般的なプログラミング言語では代入を表すために特殊な短い記号を用いるが、プログラムに慣れた人にとっては理解しやすくても、初心者にとっては意味が分かりにくいため学習に向いている表記とは言えない。図 11 は日本語 Squeak における代入命令タイルであるが、代入を「←」という矢印記号で表している。初心者にとっては「←」が何を意味しているのか、矢印の右側にあるものが代入値であり左側にあるのが代入先であるということを理解するのは難しい。

「ことだま on Squeak」では代入という概念の理解を助けるためには、その概念に近い日本語を採用して表現している。図 12 は「ことだま on Squeak」における代入文の表記である。

代入を表す日本語表記として図 12 を採用するまでには、多くの試行錯誤があった。最初に採用されたのは「～を～に代入する」という表現であるが、この表現は図 11 とあまり変わらない。「代入」という言葉は初心者にとって馴染みのある言葉ではなく、「←」と同じような意味不明な記号として捉えられる可能性があるためである。

次に採用されたのが「～を～にする」という表現であるが、これは曖昧な表現になりがちであるため不採用となった。例えば「車の向きを６０度にする。」などのように、値を変数に代入するときは、一般的に曖昧な表現にならない。ところが「車の向きをハンドルの向きにする。」などのように、変数に変数を代入するときは必ず曖昧になってしまう。この表現から、どちらが代入先でありどちらが代入値であるのかが決定できない。

以上のような検討の結果、「ことだま on Squeak」における代入表現は図 12 を採用した。この表現であれば日本語として意味が伝わり、かつ曖昧さも無く適切に意味が伝わる。

6. おわりに

本稿では「ことだま on Squeak」が持つ日本語プログラミング環境を紹介し、その環境がプログラミングの初歩概念の理解に役立つことを述べた。日本語で順次構造や分岐構造や代入文が記述されており、それらが持つ性質が日本語で記述されたプログラミングを読解することで理解できることが期待できる。

参考文献

1) 情報処理学会:日本の情報教育・情報処理教育に関する提言 2005, 情報処理学会 (2005).
2) 中川正樹, 早川栄, 玉木裕一, 曽谷俊男:日本語プログラミングの実践とその効果, 情報処理学会,Vol.35,No.10,pp.2170–2179(1994).
3) 中川正樹, 早川栄一, 並木美太郎, 高橋延匡: 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,Vol.J77-D-1,pp.364–374(1994).
4) http://www.squeakland.jp