学校教育用オブジェクト指向言語「Dolittle」の提案*

兼宗 進　　久野 靖

筑波大学大学院 経営・政策科学研究科 企業科学専攻
{kanemune, kuno}@gssm.otsuka.tsukuba.ac.jp

概要

筆者らが開発を進めているプログラミング言語「Dolittle」を紹介する。Dolittle はプロトタイプ方式を採用したオブジェクト指向言語で、初中等教育におけるプログラミング教育で活用されることを目指して設計されている。プログラミングの専門教育を受けていない教員が限られた時間内で授業を行う中で、「プログラムの楽しさを体験する」「簡単なプログラムを完成できる」「生徒の将来に害を与えない (変な癖を付けない、計算機を嫌いにならない)」ことを目標にしている。

1. はじめに

「情報社会」の現代において、次世代の社会を担う若年層に情報技術に関する適切な理解を持たせることは極めて重要である。2002 年から中学校の教科「技術・家庭」において内容の半分が情報に関するものとなり、必修化されること、および 2003 年から普通高校において新教科「情報」が選択必修という形で導入されることも、このことを見据えてのことだと考えられる [1]。

しかし、現時点で計画されているこれらの教科の学習内容を見ると、情報をさまざまな既存のソフトウェアを用いて活用したり、情報にまつわるさまざまな社会的側面を学ぶことに主眼が置かれており、情報システムの中枢部分にある計算機とその本質的な働きについて学ぶことはほとんどなさそうである。

計算機の働きについてもっとも効果的に学ぶ手段の 1 つは実際にプログラミングを体験することであるが、この選択肢は現在において極めて採用されることが少くなっている。たとえば「情報」の中でもっともコンピュータサイエンスに近いはずの科目である「情報 B」ですら、「アルゴリズムは学ぶがプログラミング言語によるプログラミングは取り上げなくてもよい」という立場を取っている [2]。

このような状況に至った背景の 1 つは、教育現場で採用されているプログラミング言語が依然として「Basic」「LOGO」などをメインとしている点にある。これらは計算機科学の現状に照らしていえばもはや 2 世代ないしそれ以上前の言語仕様であり、これらの言語を用いてプログラミングを体験しても、現代におけるソフトウェアシステムのさわりすら伺い知ることはできないし、作成したプログラムも日常接しているソフトウェア製品とは決定的に隔たったものにしか見えない。

その結果、プログラミングを体験することの価値は割り引かれることになるし、学ぶ側の動機づけも極めて低いものになりがちである。このような状況では、情報教育の一環としてプログラミングが採用されづらいのも残念ながらうなずけてしまうところである。

本研究では、このような現状を打破すべく、「現代のソフトウェアシステムのあり方を (箱庭的であっても) 体験でき」「日常使っているソフトウェ

*Dolittle – An Object-oriented Language for School Education, Susumu Kanemune and Yasushi Kuno, Graduate School of Systems Management, University of Tsukuba, Tokyo.

ア製品と (規模の差はあっても) 原理的に同じように動作するソフトウェアが作成できる」プログラミング言語および処理系を開発する。もちろん、これらが初中等レベルの生徒にとっても敷居が高くない程度の簡単さで達成できることは当然の前提となる [3]。

2. 教育におけるプログラミング言語

上記の目標を達成するため、プログラミング言語としては次のような特性を取り入れる。

1. オブジェクト指向言語であること [4]。現代のソフトウェア開発においては、既に用意されている部品を再利用することで短期間に高度な機能を持つソフトウェアが作成できるようになっている。教育用の言語であっても同じことが体験できることが必要である。そのぶん、計算機ハードウェアとのギャップは増大するが、「記述されたコード通りにプログラムが動作する」という性質が維持される限り、「計算機の動作の本質を理解する」という本来の目標がスポイルされることはないと考える。
2. プロトタイプ方式であること [5] [6]。現代において多くのオブジェクト指向言語はクラス方式であるが、クラス方式ではクラス、インスタンス、継承など理解しなければならない概念が多くなり、「敷居の低さ」の点で問題があると考える。プロトタイプ方式であれば、あるオブジェクトのコピーは元のオブジェクトの性質を引き継いでいる、という常識的なパラダイムのみで済むので、より教育用として適していると考える。
3. テキスト表現に基づくソースコード。近年、programming by example や図的プログラミングのように画面上でさまざまなオブジェクトに「直接教える」ことでプログラミングを行なうアプローチがあるが、我々が教えたいのはあくまでも「テキストとしての記号的表現」が計算機によって実行される、というモデルである。よって、テキストに基づくソースコード表現を打ち込んで動かす、というモデルは維持したい。
4. terse であること。Basic や LOGO が成功した要因の 1 つとして、文 1 つだけ打ち込んでもそれはそれで動作が観察できる、という点が挙げられよう。この利点を引き継ぐために、(常識的な長さの)1 行だけでそれなりの動作が記述でき、それがそのままメソッドとしても定義できることが必要だと考える。「クラスの中にメソッドがあり、メソッドの中に文がある」という旧来のクラス方式の言語はその点だけで既に問題外である。
5. 日本語との対応性。初等 (小学校) からの利用を考えると、英語の使用は問題外である。よって、英語の予約語等は一切ないものとする (そもそも予約語という概念は難しいので、予約語をなくす)。識別子への日本語の利用は当然のこととし、基本的な記号 (かっこ、カンマ、ピリオド等) と日本語のみで記述可能な言語とする。さらに、カンマでなく句点 (、)、かぎかっこでなく日本語の括弧 (「」) を許すこととする。語順もなるべく日本語の語順に近づける。
6. 入れ子構造を避ける。従来のプログラミング言語においては、構文の入れ子構造は当然のこととして受け入れられて来たが、初中等教育ではこれは難しい。従って、言語仕様としては複数レベルの入れ子が書けるとしても、2 レベル以上の入れ子構造を作る代りに内側の動作を別のメソッドとして分離して入れ子のレベルを押えるようなプログラミングスタイルを採れる言語とする。

さらに、実行系として次のような特性を持たせる。

1. オブジェクトが画面上でも目に見える対象物として現われ得るようにする。これをマウス等で操作すると、マウスイベントがメッセージとしてそのオブジェクトに送られるようにする。それらのメッセージを受け取るメソッドを定義するだけで、マウス操作に反応するオブジェクトがプログラムできるようにする。

これは日常操作しているGUIプログラムと本言語で作るプログラムとのギャップを小さくする効果をもたらす。さらに、通常のGUIプログラムのようなボタン、メニューといったありきたりの(決まった)動作をするもの以外のものを体験させ、なおかつその動作を自分でプログラムすることで、計算機のユーザインタフェースに関する本質的理解を深める効果が期待される。

2. LAN内でオブジェクトを共有する「場」を用意し、そこを通じてオブジェクトの参照を交換することで、複数のマシンにまたがった分散プログラムが作成できるようにする。これにより、ネットワーク、分散、CSCWといったより高度なソフトウェアのあり方に対する理解が得られるという効果が期待される。
3. さまざまな興味深い動作を実現するための「部品」(オブジェクト)を当初から十分用意し、これらを活用するスタイルが自然に身につくような環境を提供する。たとえば「本」ないし「巻物」のようなオブジェクトが最初から見えていて、これを「読んで行く」と実際に動くオブジェクトが説明ともに「貼りつけて」あって試せる、というふうに。Smalltalk-80のブラウザのような階層構造は初中等教育には難しいので、1次元に読み進めて行ける「本」や「巻物」が適している。
4. プログラムの動作している様子とその原理となるプログラムを結び付けられるようにする。例えば、画面上で動いているオブジェクトをクリックすることで、いつでもオブジェクト内の中身(プロパティやメソッドに相当するスロット)を表示することができる。その場で値やプログラムの一部を変更して動かすことにより、現象の持つ原理を発見したり、プログラムを体感的に学ぶことが可能である。

現在は、前に掲げた言語の方針に沿って仕様の検討を進めている。実行系としてはプロトタイプ処理系を作った段階であり、本項目の1〜4の実装には至っていない。将来の構想とする。

```
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！90度 回転 (青) 塗る。
白矩形 = 矩形！作る 50 右移動 (白) 塗る。
赤矩形 = 白矩形！作る 50 右移動 (赤) 塗る。
```

図 1: プログラム例 (フランス国旗)

3. 言語設計

図2に構文、図1にサンプルプログラムを示す。構文が非常にシンプルなことがDolittleの特徴である。以下に、サンプルの解説をまじえながら言語の主要な機能と特徴を解説する。

3.1 メッセージ送信

```
カメ太 = タートル！作る。
```

Dolittleはオブジェクト指向言語であり、オブジェクトへのメッセージ送信が基本となる。「！」の左はレシーバであり、右にメッセージセレクタ(メソッド名)を書く。

この例では「タートル」は広域変数名(正確には広域変数はトップレベルオブジェクト「ルート」のプロパティとして格納されている)であり、そこには予め用意されたタートルオブジェクトが格納されている。

「作る」メソッドは新しいオブジェクトを生成し、そのプロトタイプをタートルオブジェクトにする。これにより、以後このオブジェクトはタートルオブジェクトのプロパティおよびメソッド一式と同じものを予め持つかのようにふるまう。

「カメ太」はここで新たに作成する広域変数名であり、そこに新しく作ったオブジェクトを格納する。Dolittleでは変数(オブジェクトのプロパティ)は最初に値を格納した時に作られ、予め宣言する必要はない。

```
プログラム ::= (文 '。')…
文 ::= [変数 '='] 式
変数 ::= [項 '.'] 名前
式 ::= 単純式 | 送信
送信 ::= [項] '!' 電文
電文 ::= 単純式… 名前 (';' 単純式… 名前)…
括弧 ::= '(' 中置式 ')' | '(' 送信 ')'
単純式 ::= 整定数 | 文字列 | 括弧 | ブロック
ブロック ::= '[' ['|' 名前… '|'] 文 ('。' 文)… ']'
中置式 ::= 中置式 演算子 中置式 | 項
項 ::= 単純式 | 名前
```

図 2: Dolittle の構文

3.2 メッセージ送信のカスケード

```
矩形 = カメ太！50 上へ 100 右へ 50 下へ
閉じる 図形にする。
```

メッセージは任意個数の引数を持てる。引数は数値リテラルや文字列リテラルはそのまま書けるが、変数参照や一般の式は「()」で囲む必要がある。識別子が来るとそれがメッセージセレクタとして認識され、そこまでが 1 つのメッセージ送信式となる。

メッセージはオブジェクトを値として返すので、その右側に続けて引数とメッセージセレクタを書くことで、返されたオブジェクトに対するメッセージが続けて指定できる。これをカスケード (直列) 送信と呼ぶ。カスケード送信はいくつでも書くことができ、「。」でその最後を表す。

この例では次のステップで実行が行われる。

1. 「カメ太！50 上へ」が実行され、返値としてカメ太自身が返る
2. カメ太に「100 右へ」が送られ、返値としてカメ太自身が返る
3. カメ太に「50 下へ」が送られ、返値としてカメ太自身が返る
4. カメ太に「閉じる」が送られ、返値としてカメ太自身が返る
5. カメ太に「図形にする」が送られるが、このメソッドはカメ太ではなくこれまでに描かれた軌跡を新たな図形にしたものを返す
6. 新しく作られた図形オブジェクトを広域変数「矩形」に格納する

なお、図形オブジェクトはそれ自体自由に移動、変形、回転できる。

3.3 リテラルと変数の参照

```
矩形！90度 回転 (青) 塗る。
```

数字で始まるトークンは数値リテラルである。リテラルの末尾に単位をつけることができる。現在は単位は無視される (コメントとしての役割りを果たしているとは言える) が、将来的には数値オブジェクトに単位の情報を付随させて扱うことを検討している。

上の例には現れていないが、文字リテラルは文字を「""」または「『』」で囲んだものとして表す。

それ以外のリテラルは用意していないが、色などはその色名に対応する広域変数に値を格納することでプログラマの便宜を計る。たとえば、広域変数「青」には青色を表す色オブジェクトが格納されている。ここで、「()」の中に書くことでメッセージセレクタではなく変数への参照を表していることに注意されたい。

3.4 オブジェクトの複製

白矩形 = 矩形 ! 作る 50 右移動 (白) 塗る。
赤矩形 = 白矩形 ! 作る 50 右移動 (赤) 塗る。

オブジェクトの複製を行うメソッドは現在のところ用意していないが、前述の「作る」を用いることでほぼそれと同様のことができる。ただし、「作る」では元からあるオブジェクトの側がプロトタイプとなるため、そのプロパティを変更したり、それにメソッドを追加すると、新しく作られた側から (その名前のプロパティやメソッドを書き換えていない限り) 変更が観測される点に注意が必要である。このために「複製」を別途用意することも検討している。

上の例ではプロトタイプ側の変更は行っていないので、単に矩形オブジェクトを複製し、位置を移動した後で色を塗っていると考えてよい。結果として、フランスの国旗が描かれる。

3.5 メソッド定義とブロック

カメ太. 矩形 = [|x y| ! (2 * x) 上へ (x) 右へ (2 * x) 下へ 閉じる (y) 塗る]。

メソッドはオブジェクトのプロパティとしてブロックを格納することで定義する。ブロックは[…] または「…」で表し、その内側のコード列は後でブロックが評価される時に実行される。

ブロックは任意個数のパラメタを持つことができる。パラメタはブロックの先頭に「| 識別子… |」という形で指定する (指定がない場合は引数のないブロックとなる)。

パラメタはブロックの実行中だけ存在する無名のコンテキストオブジェクトのプロパティとして扱われ、初期値としてブロック評価時に渡された実引数値が格納されている。

ブロックを格納しているプロパティはメソッドとして実行可能である。その際、メッセージセレクタより前に書かれた実引数値が 1 つずつパラメタに対応する。ブロックが持つパラメタより多い場合はそれは捨てられ、少ない場合はパラメタは未定義オブジェクトを初期値として持つ。ブロック内で最後に実行された式の値がメソッドの返値となる。

3.6 複数回のメッセージ送信

カメ太 !50 (青) 矩形 ; 50 (白) 矩形 ; 50 (赤) 矩形。

メッセージ送信を表す式の途中に「;」があると、その右側は再度「 ! 」の左に書かれたレシーバへのメッセージとして扱われる。この例では、1 個目の「矩形」の返値は図形オブジェクトであるが、それは無視して次の矩形を作るメッセージを再び「カメ太」に送っている。

これにより、3 つの矩形が 3 つの色で描かれ、フランス国旗の絵が描かれることになる。

3.7 条件判断

(x>y) ! なら [...] 実行。
(x>y) ! なら [...] そうでなければ [...] 実行。

大小比較などの論理演算子は論理値オブジェクトを返す。また、ループの終了条件のように繰り返し評価する必要があるものは論理値オブジェクトを返すブロックを使って表す。制御構造は論理値オブジェクトや論理値を返すようなブロックオブジェクトへのメッセージおよびそのカスケード送信として実現する。

上の例では、メッセージ「なら」を論理値オブジェクトに送ると、論理値が真ならオブジェクト T1、偽なら F1 が返される。ブロックに「なら」を送るとブロック自身を実行し、最後の評価値の真偽に応じて同様に T1 または F1 を返す。

オブジェクト T1 は「ブロック 実行」を送られるとブロックを実行し、未定義オブジェクトを返す。「ブロック そうでなければ」を送られるとブロックを実行し、オブジェクト F2 を返す。つまり T1 は「条件が真だったので then 側へ行く」という状態を表している (F2 の意味は後述)。

オブジェクト F1 は「ブロック 実行」を送られるとブロックを実行せずに未定義オブジェクトを返す。「ブロック そうでなければ」を送られると

オブジェクトT1を返す。このT1に「ブロック 実行」が送られることで最初の条件が偽だったときに2番目のブロックが実行される。一般にF1は「まだ条件が真のものはないのでelse側へ行く」という状態を表している。

オブジェクトT1にはさらに「論理値 なら」または「ブロック なら」を送ることもでき、その(ブロックが返す)論理値が真ならオブジェクトT1、偽ならオブジェクトF1を返す。これによって次のようなif-then-else ifが使えるようになる。

```
(x>y)！なら [...] そうでなければ [(x<y)]
    なら [...] そうでなければ [...] 実行。
```

なお、2番目の条件はブロックにしなくてもよいが、その場合1番目の条件の如何に関わらず評価されてしまうことになる。

最後に、F2は「論理値 なら」、「ブロック なら」、「ブロック そうでなければ」すべてに対してF2自身を返し(ブロックは評価しない)、「ブロック 実行」に対してブロックは実行せず未定義オブジェクトを返す。すなわちF2は「既にthenの枝を実行ずみ」という状態を表している。

3.8 タイマーによる繰り返し

```
時計＝タイマー！作る 1秒 間隔 10秒 時間。
時計！「矩形！36度 回転」実行。
時計！待つ。
```

タイマーは、ブロックを一定間隔で実行するオブジェクトである。内部に実行間隔と実行時間の状態を持つ。ブロックは非同期に実行され、ブロックを起動した処理はブロックの終了を待たずに先に進む。1つのタイマーに複数の処理を頼んだときは、タイマーはそれらを直列に実行する。タイマーの実行完了を待ちたければ、タイマーのメソッド「待つ」により同期を取ることができる。

回数を「1回」とすれば、起動した処理と指定したブロックの並行実行の機能としても使うことができる(この使い方は主な目的ではなくたまたまである。並行記述のきちんとした設計はこれからの課題である)。

3.9 標準オブジェクト

言語を学ぶための題材としての役割りに加え、オブジェクト指向言語の効果を知ってもらい、また自分のプログラム作成に達成感を持ってもらうためにも、標準オブジェクトとして何を用意するかは重要な問題である。表1に現在実装または実装を予定しているオブジェクトの一覧を示す。

表1: 基本オブジェクトの一覧

オブジェクト	説明
文字列	文字列の表現と操作
数値	数値の表現と操作
論理値	論理値の表現と操作
ブロック	メソッド定義、繰り返し操作など
配列	集合データ表現
タートル	ペン先。タートルグラフィックスを描く
図形(パス)	点や線の集合
色	3原色からの生成、混ぜ合わせ演算など
画像	拡大、回転演算など
GUI部品	ボタン、テキストボックスなど

4. プロトタイプ処理系

Dolittleのプロトタイプ処理系をJava2で記述した。実行はインタプリタ方式であり、Java2の動く環境ならどこででも動かせるようになっている。執筆時点でのソースコードは約2,300行であった。言語仕様がシンプルであることと、構文解析器生成系を使用したことにより、簡潔な実装を実現している。図3に実行中の画面を示す。

5. 高校での実験

試作した処理系を使い、高校で実験授業を行った。放課後を利用し、1年生の生徒3人に授業を

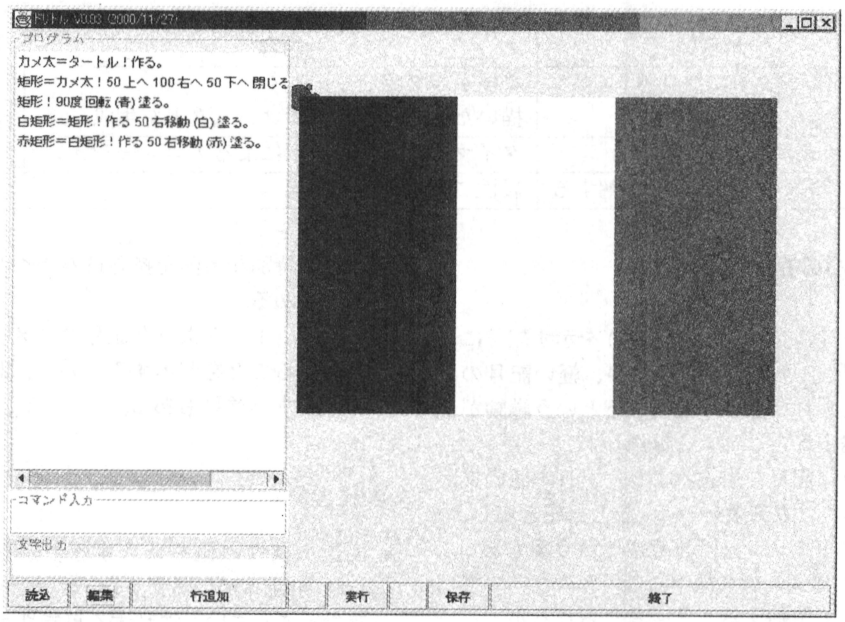

図 3: Dolittle の実行画面

行った。コースは毎週1時間ずつの3回とした。生徒のうち、1人は Visual Basic の使用経験があったが、2人はプログラミングの経験がなかった。基本的なキーボードやマウス操作には問題がなかった。表2にカリキュラムを示す。

本稿の執筆時点では1、2回目の授業が完了した段階である。そこで、ここでは2回目までの結果を報告する。

第1回

1回目はプログラミングを体験し、言語の雰囲気をつかんでもらうことを目的とした。題材としては、目に見えるオブジェクトとして、タートルグラフィックスを使用した。オブジェクトの生成として、プロトタイプからのオブジェクト生成と、描いた軌跡からの図形オブジェクト生成を扱った。結果として、「プログラムは上から順に実行されるんだ」という感想に代表されるプログラミングの基本概念と、プログラミングの楽しさを体験して終了した。

第2回

2回目はオブジェクトを意識して使うことを目的とした。題材としては、目に見えるタートルオブジェクトと図形オブジェクトに加え、目に見えないタイマーオブジェクトを導入した。結果として、「オブジェクトを生成してからオブジェクトを利用する」という基本概念が理解されていることが確認できた。また、「同じ動作を2回記述しないで済ませたい」といった、繰り返しやメソッド定義などの次のステップにつながる要求が観察された。

6. 議論

筆者らは現在の言語仕様が最終形と考えているわけではなく、さまざまな可能性を検討している。そのような議論の中から、主要なものを挙げる。

表 2: 実験授業のカリキュラム

	ねらい	内容
1回目	オブジェクトに慣れる	タートルグラフィックス 描いた軌跡から図形オブジェクトを生成
2回目	オブジェクトを意識する	タイマーオブジェクトによるアニメーション
3回目	オブジェクトの属性を理解する	属性にメソッドを定義

6.1 レシーバの指示

- メッセージの受け手 (レシーバ) を示す記号には当初「、」を使用していたが、軽い記号の割に与えられた意味が重過ぎるという議論があり、途中から「！」を採用した。
- このような記号を用いるよりも、日本語の文字を使って「カメ太を ～。」などと書き表し、助詞によってレシーバを表すという案もあった。「を」は通常の名前中には現われないので記号として予約することも考えられるが、不自然であるとの意見もある。
- 記号とする代わりに「カメ太くん ～。」「カメ太さん ～。」のようにトークンの末尾が特定文字列であることを右側からマッチングして検索する案もあった。この場合、「いかくん」(烏賊の薫製) には「いかくんさん ～。」と言えるように、末尾に複数の言い回しが使えるようにすることが重要だとの議論もあった。
- 全体として、プログラミング言語は日本語ではないことを理解してもらうためにも、日本語ふうに読めることはかえって有害である、日本語ふうにしても完璧に違和感なくすることはどのみち無理である、といった意見もある。
- 記号を避けるため、左端の識別子をレシーバーとみなすことで、「！」を書かないという案もある。ただし、あるオブジェクトが持つメソッドを他のメソッドが (最初のオブジェクトをプロトタイプとしているため) 利用するため、現在は「！～。」という書き方 (レシーバ指定部分が空であるようなメッセージ送信) を用いている (Smalltalk-80 の self に相当)。「！」を省略する場合、代りに「自分」といった偽変数を用いる必要があると思われるが、そのような特別な名前を持ち込みたくないという意見もある。
- むしろ「！」のような記号は「メッセージ送信であることを明示する」ので、あった方がよいとする意見もある。

6.2 代入

- 変数や属性の値を代入することを表すのに「=」を使っているが、(1) 数学で使う等号と重なってしまい、混乱する可能性がある、(2) メッセージ送信で統一された世界に馴染んでいない、という批判がある。
- (1) に対しては、別の記号「:=」や「←」を使えばよいという意見もあるが、(2) の回答になっていない。
- 代替案として、オブジェクトに属性を定義するメッセージを送る方式が考えられる。変数はもともと何らかのオブジェクトのプロパティなのでこれで変数の代入もカバーできる。

  カメ太！10『線の太さ』定義。

  しかしこの場合、(1) 属性名を引用する必要がある、(2) 広域変数への代入をルートオブジェクトへの代入と明示する必要がある、(3) 込み入った式の結果を代入するときに

  カメ太！(カメ子！おすすめの太さ)『線の太さ』定義。

  のように丸かっこで囲む必要がある、という問題が指摘されている。
- 引用をなくすためには、前記の案が Lisp の set だったのに対し、setq を導入して

  カメ太！10 線の太さ 定義。

  とすることが考えられる。しかしすべての引

数は評価されるという原則を破って特殊形式を導入することには異論がある。また、メッセージセレクタと引数が同じ形をしているため、右から構文解析をする必要があり、書き違いの発見が困難になるという問題もある。

- かっこのレベルを押えるため、値とプロパティを格納するオブジェクトの順序を交換する案もある。

  カメ子！おすすめの太さ (カメ太)『線の太さ』定義。

  しかし「カメ太」を丸かっこで囲むことに変わりはない上に、self を表す偽変数が必要となるという問題がある。

6.3 数式

- 通常の数式は、そのままではメッセージ送信の枠組みの中に取り込むことができない。丸かっこの中には (1) メッセージ送信、(2) 通常の中置記法、のいずれかが書けることとし (「！」の有無で判別可能)、数式は

  x = (y + 1)。

  のように中置記法で記述するようにしている。変数の参照もこの特別な場合に相当する。
- 日本語として「1 と 2 を足して 4 を掛ける」のように読み下せる逆ポーランド記法も検討したが、高学年の複雑な数式を変換して記述する手間を考えて中置記法を可能にした。逆ポーランド記法を採用した場合、本当は「1 2 + 4 *」と書きたいところだが、現在の文法では「1！2 足す 4 掛ける」のように「！」を入れて記述することになる。

6.4 日本語の扱い

- 実験授業などでの経験から、標準オブジェクト名やメソッド名、プロパティ名などの識別子が日本語であること、記号も日本語の引用かっこや読点を使ったことは、言語の親しみ易さを増す効果があったと考えている。
- 日本語での入力につきものの 16 ビット文字 (いわゆる全角) と 8 ビット文字 (いわゆる半角) の問題についても、できるだけ両者を区別しない (同一の文字として扱う) ようにすることは不要なつまづきを減少させる効果があった。
- ただし、「。」と「.」を同一視し (かな漢字変換の設定で文末を「.」にしている人には使いやすい)、なおかつ「.」と「.」を同一視させるべきかどうかについては議論がある。現在は「。」と「.」は別のものであり、「.」と「.」を同一視している。
- わかち書きを避けたり、助詞などによる語の役割りの判別を導入することで、より日本語らしくしたいとする意見もある。その反面、プログラミング言語はあくまでも人工言語であることを学んで欲しいのであり、むやみに日本語と混同しがちな言語を学ばせることは学習者の将来にとって有害だとの意見もある。

7. まとめ

教育用に適したオブジェクト指向言語を考案し、実装と実験を行った。このような言語を通じて、初中等教育において、すべての生徒が自分のレベルと興味に応じてプログラミングを体験し、計算機に関する理解を深められることを目標として改良を進めて行きたい。また、現時点では中等教育レベルを想定して言語設計を行っているが、今後はさらに検討を進め、初等教育における活用の可能性も考えて行きたい。

謝辞

本研究は、情報処理振興事業協会 (IPA) の平成 12 年度未踏ソフトウェア創造事業の補助を受けています。また、電気通信大学の竹内郁雄氏、アーマットの御手洗理英氏、SLagoon の中谷多哉子氏、日本電気の福井眞吾氏から有用なコメントをいただきました。筑波大学附属高校の矢野一幸先生には実験に協力をいただきました。ここに感謝いた

します。

参考文献

[1] 情報処理学会初中等情報教育委員会 WG: 高等学校普通教科「情報」試作教科書, http://www.ics.teikyo-u.ac.jp/InformationStudy/

[2] 文部省 : 学習指導要領, http://www.monbu.go.jp/news/00000317/

[3] 兼宗, 久野 : 学校教育用オブジェクト指向言語/環境の構想について, 情報教育シンポジウム (SSS2000) 予稿集, 情報処理学会, 2000

[4] A.Goldberg and D.Robson: *Smalltalk-80: The Language and Its Implementation*, Addison−Wesley, 1983

[5] D.Ungar and R.Smith: *Self: The Power of Simplicity*, ACM OOPSLA'87, 1987, pp.227−242

[6] ECMA: *ECMAScript Language Specification*, http://www.ecma.ch/ecma1/stand/ecma-262.htm