学校教育用オブジェクト指向言語「ドリトル」の設計と実装

兼宗 進[†] 御手洗 理英[†,††] 中谷 多哉子[†††]
福井 眞吾[†] 久野 靖[†]

情報社会の急速な発展にともない，初中等教育の中で情報の比重が高まっている．計算機の働きを最も効果的に学ぶ手段の 1 つはプログラミングを体験することであるが，教育現場では Basic や Logo といった数世代前の言語が使われることが多く，現代のソフトウェアシステムの理解につながらないという問題が存在する．本稿では，初中等教育での利用が可能なプログラミング言語「ドリトル」およびその実行系の設計と実装について述べる．ドリトルはオブジェクト指向言語であり，あらかじめ用意された各種のオブジェクトを活用した教育を可能とする一方，Self 言語と同様のプロトタイプ方式の採用により，クラスや継承などの高度な抽象概念の理解を不要にしている．その他，変数や命令語などの識別子と記号が日本語文字で統一されている，メソッドを属性と統合的に扱えるといった特徴を持つ．処理系は Java2 で書かれたインタプリタとして実装し，教育現場のさまざまな環境で動作できるようにした．

Dolittle: An Object-oriented Language Aimed at K12 Education

Susumu Kanemune,[†] Rie Mitarai,[†,††] Takako Nakatani,[†††]
Shingo Fukui[†] and Yasushi Kuno[†]

In the IT revolution, IT education is becoming more important in school education. Programming is an effective way for learning computers. However, many teachers use old languages like Basic and Logo, so students can't understand modern software systems. This paper describes design and implementation of the programming language "Dolittle". Dolittle is an object-oriented language aimed at school education. Incorporating prototype-based object system like Self, Dolittle requires less knowledge of abstract concept like classes and inheritances. Students can learn it easily, thanks to predefined objects and familiar Japanese identifiers and symbols. We implemented Dolittle interpreter by Java2, so it can run in many educational environments.

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きわめて重要である．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[6]．

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んだりする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最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採用されることがきわめて少くなっている．たとえば「情報」の中で最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」という立場をとっている[9]．

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 や 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

† 筑波大学大学院経営・政策科学研究科
Graduate School of Systems Management, University of Tsukuba
†† 株式会社アーマット
Armat Corporation
††† 有限会社エス・ラグーン
SLagoon Co., Ltd.

ソフトウェアシステムのさわりすらうかが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きわ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にくいのも残念ながらうなずけてしまうところである．

我々は，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について研究している．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[7]．

2. 教育におけるプログラミング言語

2.1 求められる特徴

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，すで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,「記述されたコードどお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失わ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[2),12)]．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，例示によるプログラミング（programming by example）や図的プログラミングのように，画面上で図形的な操作を行うことでプログラミングを行うアプローチがある．しかし，今回は「計算機の動作原理を体感する」ことを目的とするため，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することにした．

(4) 簡潔であること．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，という点があげられる．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すでに問題外である．

(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，英語の予約語などはいっさいないものとする（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，ピリオドなど）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) 階層的な構文を避ける．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きたが，変数の有効範囲などが階層的になり，初中等教育では理解が難しい．したがって，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わ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おさえるようなプログラミングスタイルをとれる言語とする．

2.2 既存言語の考察

言語の設計にあたり，既存の教育用言語を検討した．

(1) Logo[10)] は教育用に設計され，現在も教育現場で利用されている言語である．画面に視覚的な操作対象（タートル）を置き，動かした軌跡を画面に残すことにより，ユーザが自分の行った操作を目で確認しながらプログラムを作ることができる．制御構造は再帰を基本とし，データ構造としてリストが用意されている．Logo は少ない概念でプログラムを書けるという利点を

持つが，一方で学習者が再帰やリストという高度な概念を学習する必要があり，対話的なタートルグラフィックス以降の学習に結び付けるのが難しいという問題がある．今回の言語では，入門用の題材としてタートルグラフィックスを利用することにした．ただし，従来のタートルグラフィックスでは，操作対象はタートルであり，軌跡は単に画面上の絵として扱われるが，今回の言語では描かれた図形を新たなオブジェクトとして扱えるように拡張を行っている．

(2) Smalltalk-80[4] は Kay らにより教育で使われることを視野に入れて設計されたオブジェクト指向言語である[8]．言語全体がオブジェクト指向で統一され，メッセージ送信によって動くモデルは今回の言語の参考にした．ただし，クラス階層の理解とクラス定義は初心者がプログラムを作るうえで負担が大きいという判断から，今回の言語ではクラス方式のオブジェクト指向を採用しなかった．

(3) Cocoa[1] に代表される例示によるプログラミングは，特別な文法を覚えなくても初心者がプログラミングを体験できる利点がある．しかし，「画面上のマウスなどによる操作を計算機が学習する」というモデルは，我々の目的の 1 つである「生徒が計算機の動作原理を体感する」効果が薄くなると判断したため，採用しなかった．

(4) 図形プログラミングのうち，LEGO Mind-Storms[11] はロボットのような対象を順に操作する手順を記述することに適しており，国内でも教育用に使われはじめている．しかし，今回の言語では手続き的な操作に限らないオブジェクト間のメッセージを扱いたかったため，テキストによる記述を採用することにした．

3. 言語設計

図 1 に構文，**図 2** にサンプルプログラムを示す．構文は非常にシンプルである．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。

ドリトル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（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，そこにはあらかじ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あらかじめ持つかのように振舞う．

「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．ドリトルでは変数（オブジェクトのプロパティ）は最初に値を格納したときに作られ，あらかじ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 ::= 数値定数 | 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 1 ドリトルの構文
Fig. 1 Syntax of Dolittle.

```
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！(青) 塗る。
白矩形 = 矩形！複製 50 右移動 (白) 塗る。
赤矩形 = 白矩形！複製 50 右移動 (赤) 塗る。
```

図 2 プログラム例（フランス国旗）
Fig. 2 Program sample (France flag).

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．引数のうち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！(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を果たしているとはいえる）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図る．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！複製 50 右移動 (白) 塗る。
赤矩形 ＝ 白矩形！複製 50 右移動 (赤) 塗る。

「複製」メソッドはレシーバオブジェクトの複製を返す．複製されたオブジェクトは元のオブジェクトと同じプロパティ，メソッド，プロトタイプを持つ．上の例では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．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ときに実行される．

ブロックは任意個数のパラメータを持つことができる．パラメー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）．

パラメー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 つずつパラメータに対応する．ブロックが持つパラメータより多い場合はそれは捨てられ，少ない場合はパラメー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評価された式の値がメソッドの返値となる．

3.6 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．制御構造は，論理値オブジェクト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，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 True または False を返す．

オブジェクト True 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 Done を返す．つまり True 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（Done の意味は後述）．

オブジェクト False 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 True を返す．この True 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 False 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．

☆ 条件部はブロックに入れずに条件式を直接書くことが可能だが，実行時に必ず評価される点に注意が必要である．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場合や，条件分岐の else-if 部の条件のように必要になるときまで評価したくない場合があるので，標準ではブロックの使用を推奨している．

オブジェクト True にはさらに「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 True，偽ならオブジェクト False 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。

最後に，Done は「ブロック なら」および「ブロック そうでなければ」に対して Done 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 Done は「すでに then の枝を実行ずみ」という状態を表している．

3.7 条件による繰返し

[x<100]！の間 [カメ太！1 歩く。x= (x+1)] 実行。

条件式を含むブロック B1 にメッセージ「の間」を送ると，内部にブロックを包含するオブジェクトが返される．このオブジェクトに繰り返し実行されるコードを含むブロック B2 を引数としてメッセージ「実行」が送られると，オブジェクトは B1 を実行し，その値が論理値の真の間だけ B2 を繰り返し実行する．

3.8 タイマーによる繰返し

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」により同期をとることができる．

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．

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** に現時点での標準オブジェクトの一覧を示す．この中で，数値，論理値，文字列，ブロックは言語の基本部分を構成する基本オブジェクトであり，それ以外はシステムが用意する標準オブジェクトである．

表 1 標準オブジェクトの一覧
Table 1 List of basic objects.

オブジェクト	説明
数値	数値を表し，各種の演算を実現する 内部では 64 bit の浮動小数点で値を保持する
論理値	論理値を表し，論理演算を実現する
文字列	文字列を表し，連結等の操作を実現する
ブロック	メソッドの定義に使われるほか， 繰返しなどの制御構造を実現する
配列	集合データを表現する
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）	点や線の集合を表す
色	3 原色からの生成や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

表 2 ドリトル処理系のソースコード行数
Table 2 Source code lines of Dolittle.

字句/構文記述（SableCC）	150 行
Java 2 コード	2,600 行

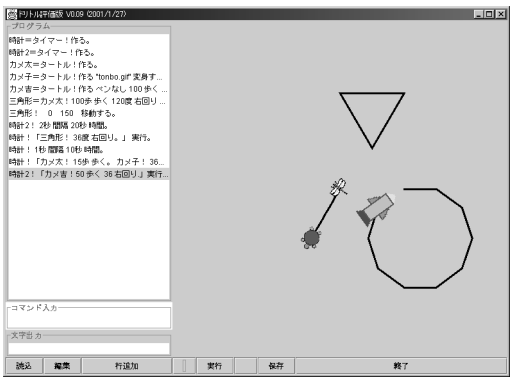


図 3 ドリトルの実行画面
Fig. 3 Execution of Dolittle.

4. 処 理 系

ドリトルの処理系は Java 2[5] により記述されており，Java 2 が動くさまざまな環境で動作する．字句解析，構文解析は SableCC[3] による生成コードを利用し，これに加えて意味解析，インタプリタ，ユーザインタフェース，および標準オブジェクト群を実装している．本稿執筆時点でのソースコード行数は**表 2** のとおりである．

ユーザインタフェース部分（**図 3**）はファイルやキーボードからのプログラムコードの読み込み，プログラムの画面上での編集，ファイルへの保存，および実行機能を提供する．プログラムの実行が指示されると，現在読み込まれているプログラムを対象として字句解析，構文解析が起動され，エラーなしに抽象構文木が

生成された場合にインタプリタによる実行に進む．

インタプリタは抽象構文木を直接たどりながらドリトルの動作を実行する．ドリトルはオブジェクト指向言語であり，プロパティの参照/設定，メソッドの起動など少数の基本動作を除き，ほんとんどの動作はオブジェクトの動作を呼び出すことを通じて行われる．

処理系内部では，ドリトルのオブジェクトは Java 言語のクラス OxObject およびそのサブクラスとして実装されている．クラス OxObject のインスタンスはハッシュ表を保持しており，ここに各オブジェクトのプロパティ値を保管する．また OxObject およびその各サブクラスごとに 1 つずつ別のハッシュ表があり，そこにそのクラスで利用可能な組み込みメソッド（Java で記述されたメソッド）の情報が（Java のリフレクション機能のメソッドオブジェクトとして）保持されている．

システムにただ 1 つ存在するルートオブジェクトを除くすべての OxObject およびそのサブクラスのインスタンスは，インスタンス変数 parent に自身のプロトタイプ（親）オブジェクトへの参照を保持しており，プロパティを検索して自分のハッシュ表に見つからない場合にはプロトタイプに検索を委譲する．これにより，自分からルートまでのプロトタイプ連鎖のどこかでプロパティが定義されていればそれが参照される．プロパティに値を設定するときはこのような検索は行わず，直接自分のハッシュ表に登録する．これにより，親側でのプロパティ変更はすべての子供に波及するが，子供側での変更は親や兄弟に影響することはないという性質が実現される．

クラス OxObject では invoke() というメソッドを実装しており，メソッド名（文字列）と引数リスト（OxObject の配列）を指定してこのメソッドを呼び出すことで各オブジェクトのメソッドが呼び出せる．invoke() ではまずメソッド名でプロパティを検索し，ブロック値（OxBlock のインスタンス）が取り出せれば，それをドリトルで記述されたメソッドとして実行する（OxBlock のメソッド invoke() を実行する）．見つからなかった場合には，組み込みメソッドの表を検索し，見つかったメソッドをリフレクション機能を通じて直接呼び出す．いずれも見つからない場合はその呼び出しは失敗し，未定義オブジェクトが返される．

OxBlock はブロック内部のコードの抽象構文木を保持しており，invoke() はインタプリタによりこの抽象構文木を解釈実行する．OxBlock は生成されたときのコンテキストを記憶しており，これを通じて変数を検索するため，ブロックの外側で定義されている変数を参照/更新できる．ただし，ブロックがメソッドとして起動されるときはこのコンテキストは使用されず，そのメソッドを保持しているオブジェクトがコンテキストとして使用される（メソッドがそのオブジェクトの変数群を読み書きするため）．

表 3 実験授業のカリキュラム
Table 3 Curriculum of experimental lecture.

授業	ねらい	内容
1	オブジェクトに慣れる	オブジェクトの生成 タートルグラフィックス メッセージ送信とカスケード 図形オブジェクトの生成
2	オブジェクトを意識したプログラム	複数のオブジェクト タイマーによるアニメーション
3	メソッドの定義	属性にメソッドを定義 作品作り

5. 評　　価

開発した処理系を使い，ドリトルが想定しているユーザである生徒と教員を対象にした評価を行った．評価の基準として「計算機の動作原理を体感できる」「プログラミングの楽しさを体験できる」「言語を数時間で習得できる」「作品としてのプログラムを作れる」か否かを主に検討した．

5.1 高校での実験授業

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 人が参加した．コースは 1 時間を目安とし，1 週間程度の間隔で 3 回行った．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．また，全員がワープロや電子メールの使用経験があり，基本的なキーボードやマウス操作には問題がなかった．実施したカリキュラムを**表 3** に，使用したテキストの例を付録 A.4 に示す．授業の中は，次のように進めた．

(1) その日のテキストを配る．
(2) 講師がプロジェクタでサンプルプログラムを動かしながら，新しい機能を説明する．
(3) 生徒が手元のノート PC に例題を打ち込み，実行する．
(4) 例題を修正する．
(5) 課題のプログラムに挑戦する．

評価は授業中の生徒の観察とアンケート，作品プログラムから行った．

計算機の動作原理については，「プログラムは上から順に実行されることが分かった」「同じ動作を 2 回記述しないで済ませるのは便利だと感じた」といった感想があり，今までブラックボックスだった計算機の動

表 4 教員の内訳
Table 4 Affiliation of teachers.

学校	人数（うち未経験者）
小学校	6（3）
中学校	7（2）
高校	2（0）
大学	2（0）
その他	3（0）

作原理を体感できたことを確認した．

プログラミングの楽しさについては，「自分で考えたことが，うまく画面に現れるととても嬉しかった」「どんなことをしようか決めてからプログラムを組むのはパズルのようで面白かった」といった感想から，プログラミングの楽しさを体験できたことが確認された．

言語の習得については，3 時間の授業の中で，メッセージ送信，オブジェクト生成，メソッド定義が，実際に作品を作るレベルまで理解できたことを確認した．

作品としては，タートルグラフィックを利用した複数オブジェクトのアニメーションプログラム作品を全員が作ることができた（付録 A.5，A.6）．

最終的に作られたプログラムの規模は，最大の作品で 27 行であった．その中には 10 個の明示的なオブジェクト生成，6 個のメソッド定義，65 回のメッセージ送信が含まれていた．

5.2 教員による評価

情報教育に関心を持つ教員に呼びかけて，ドリトルの体験実習を行った．参加者 20 人の内訳を**表 4** に示す．参加者のうち，15 人はプログラミングの経験があり，5 人は経験がなかった．経験者の使っている言語は Logo，Basic，Pascal などであり，日常的にオブジェクト指向言語を使っている教員はいなかった．

研修では 30 分で文法とオブジェクト指向の考え方を説明し，60 分間の実習を行った．テキストには高校の実験授業で使った 3 時間分の資料を配布し，時間が短かったため，テキストの例題を入力しながら言語を体験することを目的にした．実際には，途中から多くの受講者が自由課題にチャレンジした．

終了後の感想やアンケートでは，はじめてプログラミングを体験した教員全員から，「自分でプログラムを作れるのが面白かった」という意見が出された．プログラミング経験のある教員からは，「プログラムを簡潔に記述できる」「小学生でも簡単に使えると思う」という感想が聞かれたが，数人から「オブジェクトの考え方に戸惑いを感じた」という指摘があった．実習を観察した結果では，Basic や Logo などオブジェクトの存在しない言語に慣れている教員の場合，当初はレシーバにメッセージを送る考え方に戸惑うケースが観察された．しかし，いずれの場合も 30 分程度で違和感なく使えるようになったことを確認した．

6. 議　　論

本章では，自分たちで使った経験と生徒や教員に教えた経験に基づいてドリトルの特徴について議論する．

6.1 メッセージ送信のカスケード

ドリトルのプログラムでは，メッセージ送信で返されるオブジェクトをレシーバにして続くメッセージを送るカスケード送信を多用する．この結果，プログラムは一連のメッセージ送信の列が横に並ぶ形になる．これは次の 3 つの点で効果があった．

- レシーバを何度も書く必要がないため，プログラムの記述が簡潔になる．
- ひと固まりの仕事をするメッセージ列が一目で分かる．
- 通常の言語に比べて行数が 1/3 程度に短くなるため，画面上でプログラムの広い範囲を見渡せる．

多くのメソッドはレシーバ自身を返すが，タートルオブジェクトの「図形にする」など異なる種類のオブジェクトを返すメソッドが存在する．今回の例題では扱わなかったが，カスケード列の中で異なる種類のオブジェクトにメッセージ送信が行われる場合，混乱を招く可能性があるため，今後の検討が必要である．

6.2 レシーバの指定

レシーバは「！」という記号を使って明示する形で記述する．今回の高校生と教員を対象にした実験では，特にこの記号は問題にならなかった．

しかし，「記号に込められた意味が重すぎる」という意見や，逆に「記号を省略しても可読性が下がらない」という意見がある．そのため，左端の識別子をレシーバと見なすことで「！」を省略可能にしたバージョンを試作した（付録 A.2）．今後，比較や評価を行う予定である．

「！」の省略を許す場合には，次のような検討が必要になる．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）．このような場合には，「！」を省略することはできない．

仮にメソッドの先頭で「！」の省略を許す場合には，代わりに「自分」といった偽変数を用いることが考えられるが，特別な予約語を意識してプログラミングを行う必要が生じるという問題がある．

```
class name NewTurtle
superclass Turtle
instance variable names
   step
instance methods
   newstep: s
     step←s
   run: n
     self forward: (n ∗ step)
```

図 4 Smalltalk-80 でのクラス定義例
Fig. 4 Class definition in Smalltalk-80.

6.3 プロトタイプ方式の採用

クラスを用いず，プロトタイプオブジェクトの複製によりオブジェクトを生成する方式は，次の点で効果があった．

- クラス定義が不要である．
- 複雑なクラス階層を理解する必要がない．
- 複製というオブジェクトの生成モデルは単純で理解しやすい．

比較のために，Smalltalk-80 とドリトルのコードを対比する．ここでは「指定した倍率だけ大股で歩く」動作をする「走る（run）」というメソッドを持つ新しいタートルを作る場面を考える．

Smalltalk-80（**図 4**）のプログラムでは，既存のタートルクラスから，新しいクラスを定義している．実際には，このコードのほかにオブジェクトを生成するコードが必要であり，1 つのオブジェクトを作るために，クラスを定義してオブジェクトを生成する 10 行以上のコードが必要になる．また，構文中には「class」「superclass」「instance variable」「instance methods」「self」などの多数の予約語が出現するため，学習時にこれらの理解が必要になる．クラスは目に見えない抽象概念であり，さらにそれらの階層関係を理解しなければならないことから，初心者にとってオブジェクトを拡張することは難しい作業になる．

一方，ドリトル（**図 5**）ではプロトタイプオブジェクトを複製してオブジェクトを作り，それに必要なメソッドを定義する．複製という分かりやすい操作でオブジェクトを生成し，実際のオブジェクト生成を含めて数行で記述できるのが特徴である．

実際，今回評価として行った実験授業においては，高校生の 3 時間，教員の 1.5 時間という限られた時間の中でオブジェクトの概念を扱ったが，これはプロトタイプ方式を採用したから可能であったと考えている．仮にクラス方式を採用した場合は，授業の最初にクラス，継承，オブジェクトの概念とそれらの関係を説明する必要があり，筆者らの経験から判断して，大学生

```
カメ太＝タートル！作る。
カメ太：歩幅＝「|s|step ＝ (s)」。
カメ太：走る＝「|n|！(n∗step) 歩く」。
```

図 5 ドリトルでのオブジェクト定義例
Fig. 5 Object definition in Dolittle.

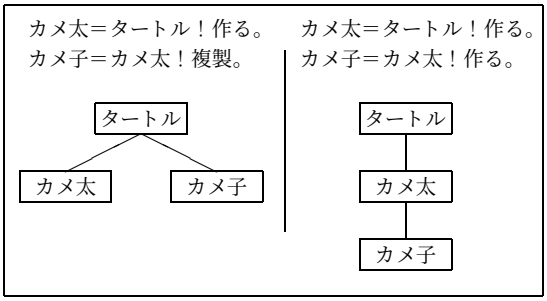


図 6 オブジェクトの生成と複製
Fig. 6 Creation and copying of objects.

が相手であっても数時間で理解してもらうことは困難であったと考えている．

6.4 オブジェクトの複製

プロトタイプ方式によるオブジェクトの複製は簡明なオブジェクト生成を可能にするが，一方，次の点に検討が必要であると考えている．オブジェクトを複製するメソッドとして,「複製」と「作る」がある.「複製」は親リンクを含めて元のオブジェクトを複製する．一方,「作る」は新たなオブジェクトが元のオブジェクトをプロトタイプとした親リンクを形成するため，元のオブジェクトにプロパティやメソッドを追加すると，新たなオブジェクトからもそれらが観測される（**図 6**）．今回の実験授業では問題にならなかったが，ユーザが意図しないオブジェクト階層が作られる可能性があるため，これらの違いを理解して使い分ける点について今後の検討が必要である．

6.5 中置記法と変数の参照

ドリトルでは予約語を定めず，変数を表す記号も用意していない．識別子は，プログラム中に出現する位置に応じて，変数ないしメッセージセレクタとして解釈される.「！」記号の左側の識別子はレシーバである．それ以外の識別子が出現するとメッセージセレクタになる．

数式などの中置演算子を丸かっこ内およびブロック内に単独で記述できるようにした．メッセージ式と中置式の区別は構文から判断する．

これらの設計は，次のような効果があった．

- 予約語が存在しないことで，ユーザが「使っていけない言葉」を意識せずプログラミングを行うことができる．

- 変数を表す記号が存在しないことで，キーボードからの記号の入力を減らすことができる．
- 数式や比較式の中で，変数をそのまま書ける．

メッセージ引数に変数を記述する場合は，中置演算子と同様の扱いとして，丸かっこで囲み，メッセージセレクタと区別する．今回の高校生と教員を対象にした実験では，特にこの記法は問題にならなかった．

6.6 メソッドの定義

メソッドはブロックをオブジェクトのプロパティに代入することにより定義する．これは次のような効果があった．

- プロパティに代入する形でメソッドの定義が簡単に行える．
- メッセージ送信のカスケードと組み合せることで，多くの場合メソッドを 1 行で記述できる．
- 特別な構文が不要．代入やブロックという基本的な構文を組み合せてメソッドを定義できる．

ブロックに「実行」メッセージが送られた場合と，ブロックをメソッドとして起動した場合は，ブロックを実行するコンテキストが異なる．これは混乱を招くのではないかという意見があったが，実際に生徒や教員の授業やプログラムを観察した限りでは，このような違いが問題になる場面には遭遇しなかった．

6.7 タイマーと並行プログラミング

タイマーオブジェクトの導入は次の効果があった．

- オブジェクトのアニメーションを実現できることは，目に見えるオブジェクトの操作の幅を広げ，生徒の動機付けに役立った．
- 複数のオブジェクトを同時に動作させることができ，並行プログラミングを体験することにつながった．

タイマーはブロックを非同期で実行させるため，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．高校生を対象にした実験授業では，ブロックの実行が終わらないうちに，ブロックの中で描いた図形をブロックを起動した処理から使おうとしたため，生徒が混乱する現象が観察された．そのため，ブロックの実行を待ち合わせるためのメソッド「待つ」をタイマーに導入したが，並行プログラミングを適切に理解してもらうための体系的なオブジェクト構成や教育方法については今後の課題である．

6.8 日本語によるプログラム

構文に日本語を許すことは，次の効果があった．

- 標準オブジェクト名やプロパティ名，メソッド名などの識別子が日本語であること，そして記号に日本語の引用かっこや読点を使えるようにしたことでは，親しみやすさが増し，英語が未修得の生徒でも使える効果があった．
- 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）ようにすることで不要なつまずきを減少させる効果があった．

表 5 同等に扱われる記号の例
Table 5 The symbols treated identically.

記号	用途
[〔 「	ブロックの開始
] 〕 」	ブロックの終了
“ ” “ 『	文字列の開始
” ” 』	文字列の終了
. . 。	文末および小数点

一方，どの記号を同一として扱うかには検討が必要なものがある．たとえば，現在は文末の記号として「。」と「.」を同一視しているが，小数点の表現に「。」を使ってよいかは異論がある．**表 5** に，現在同等に扱っている記号の例を示す．このほかにも，英字やカタカナなど，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）に同じ文字がある場合には，対応する文字は同じものとして扱っている（リテラルの中にある場合は除く）．

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，実装と評価を行った．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いきたい．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いきたい．

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ました．また，電気通信大学の竹内郁雄氏から有用なコメントをいただきました．実験授業に協力いただいた筑波大学附属高等学校の矢野先生，生徒の皆様，ドリトルの評価に協力いただいた兵庫県教育工学研究会をはじめとする全国の教員の皆様に感謝いたします．

参 考 文 献

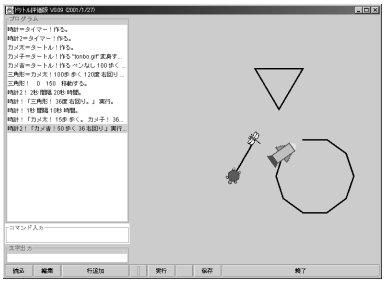
1) Cypher, A. and Smith, D.: KidSim: End user programming of simulations, *CHI′95*, pp.27–34 (1995).

2) ECMA: ECMAScript Language Specification. http://www.ecma.ch/ecma1/stand/ecma-262.htm
3) Gagnon, E.: SableCC, An Object-Oriented Compiler Framework, Master's Thesis, McGill University (1998).
4) Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley (1983).
5) Gosling, J., Joy, B. and Steele, G.: *The Java Language Specification*, Addison-Wesley (1996).
6) 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 (1998). http://www.ics.teikyo-u.ac.jp/nformationStudy/
7) 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 回プログラミングシンポジウム，pp.11–20 (2001).
8) Kay, A.: Learning vs. Teaching with Educational Technologies, *EDUCOM Bulletin*, pp.16–20 (1992).
9) 文部科学省：学習指導要領 (1999). http://www.mext.go.jp/a_menu/shotou/youryou/index.htm
10) Papert, S.: *Mindstorms : children, computers, and powerful ideas*, Basic Books (1980).
11) The LEGO Group: LEGO MINDSTORMS. http://mindstorms.lego.com/
12) Ungar, D. and Smith, R.: Self: The Power of Simplicity, *OOPSLA'87*, pp.227–242 (1987).

付録 ドリトルのプログラム例とテキスト例

A.1 アニメーションの例

図 7 は図 3 の画面で用いたプログラムである．実行すると，最初に「カメ」「トンボ」「ロケット」という 3 個のタートルオブジェクトを生成し，タートルの軌跡から「三角形」という図形オブジェクトを生成する．続いてこれら 4 個のオブジェクトを,「時計 1」「時計 2」という 2 個のタイマーオブジェクトを使って操作する．時計 1 は引数で渡されたブロックオブジェクトを，1 秒間隔で 10 秒間（つまり 1 秒間隔で 10 回）実行する．時計 2 は引数で渡されたブロックオブジェクトを，2 秒間隔で 20 秒間（つまり 2 秒間隔で 10 回）実行する．タイマーオブジェクトは非同期に実行されるので，時計 1 と時計 2 は並行に実行される．また，1 つのタイマー（ここでは時計 1）の実行中に続けて実行メッセージが送られた場合には，それらを直列に実行する．すなわち，時計 1 は三角形を 36 度ずつ回転させる操作を行った後，カメとトンボの操作を開始する．

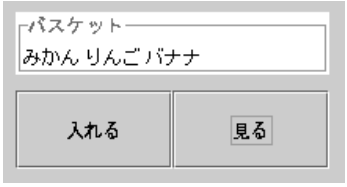


```
トンボ＝タートル！作る “tonbo.gif” 変身する。
ロケット＝タートル！作る “rocket.gif” 変身する。
カメ＝タートル！作る。
三角形＝カメ！100歩 歩く 120度 右回り 100歩 歩く 閉じる
        図形にする。
時計 1 ＝タイマー！作る 1秒 間隔 10秒 時間。
時計 1！「三角形！ 36度 右回り」 実行。
時計 1！「カメ！ 15歩 歩く。トンボ！ 36度 右回り」 実行。
時計 2 ＝タイマー！作る 2秒 間隔 20秒 時間。
時計 2！「ロケット！50歩 歩く 36度 右回り」実行。
```

図 7 タイマーオブジェクトを使ったアニメーション
Fig. 7 Timer animation sample.

```
トンボ＝タートル 作る “tonbo.gif” 変身する。
ロケット＝タートル 作る “rocket.gif” 変身する。
カメ＝タートル 作る。
三角形＝カメ 100歩 歩く 120度 右回り 100歩 歩く 閉じる
        図形にする。
時計 1 ＝タイマー 作る 1秒 間隔 10秒 時間。
時計 1「三角形 36度 右回り」 実行。
時計 1「カメ 15歩 歩く。トンボ 36度 右回り」 実行。
時計 2 ＝タイマー 作る 2秒 間隔 20秒 時間。
時計 2「ロケット 50歩 歩く 36度 右回り」実行。
```

図 8 レシーバを示す記号を省略した例
Fig. 8 No exclamation symbol sample.




```
バスケット＝配列！作る。
テキスト＝フィールド！作る “バスケット” タイトル。
ボタン 1 ＝ボタン！“入れる” 作る。
ボタン 1：動作＝「バスケット！(テキスト！読む) 入れる」。
ボタン 2 ＝ボタン！“見る” 作る。
ボタン 2：動作＝「テキスト！(バスケット！見る) 書く」。
```

図 9 GUI 部品の例
Fig. 9 GUI objects sample.

A.2 レシーバを示す記号を省略した構文の例

6.2 節で議論したように，レシーバを示す記号「！」を省略するバージョンを試作した．**図 8** に，図 7 のプログラムから「！」を省略した例を示す．文の左端に出現する識別子がレシーバとして解釈される．

2000.11.16 筑波大学ドリトル開発チーム

「ドリトル」を使ってみよう

1. はじめに

みなさんはコンピューターのプログラムを作ったことがありますか? みなさんの回りでも、ゲームソフトや携帯電話など、いろいろなものがプログラムで動いていますよね。
今日紹介するのは、「学校のクラブや授業で楽しくプログラムを作れる」ことを目標に筑波大学で作っている「ドリトル」という名前のプログラミング言語」です。
ドリトルは開発中で、これからいろいろな面白いアイデアを取り込んで発展していく予定です。ぜひ使ってみて感想などを聞かせてください。

2. 図形を描く

ドリトルでは、「もの」(図形や生き物など) にお願いする形でプログラムを書いていきます。オブジェクト指向という考え方です。「○○くん、△△してよ。」という感じです。ドリトルではこれを「○○ ！ △△。」と書きます。

プログラムを見てみましょう。

```
( 例 1：sample1.txt )
カメ太=タートル！作る。          …  (1)
カメ太！ペンなし。               …  (2)
カメ太！200 歩   歩く。          …  (3)
カメ太！120 度   右回り。        …  (4)
カメ太！ペンあり。               …  (5)
カメ太！100 歩   歩く。
カメ太！120 度   右回り。
カメ太！100 歩   歩く。
カメ太！閉じる。                 …  (6)
三角=カメ太！図形にする。        …  (7)
三角！(青)   塗る。               …  (8)
三角！0    200   移動。          …  (9)
```

①(1) の「カメ太=タートル！作る。」でタートル (カメオブジェクト) を 1 匹作り、「カメ太」という名前にする。
カメ太は、画面の真中にいて、右を向いています。
②カメ太はしっぽにペンをぶらさげているので、歩くと足跡が残ります。(2) の「ペンなし」でペンがないようにしています。
③カメ太は前に歩けます。(3) の「200 歩　歩く」で、200 歩あるき、(4) の「120 度　右回り」で 120 度右回転する。ペンを持っていないので、歩いても何も描かれません。最後に (5) の「ペンあり」でペンを持たせてやりましょう。
④100 歩前進、120 度右回りに回転、100 歩前進のように動き回る。今度はペンを持っているので、画面に歩いた跡が残ります。
⑤(6) の「閉じる」で図形を閉じます。
⑥(7) でカメ太は、自分の足跡を「三角」という名前の図形にしています。
(足跡が図形になるなんておもしろいですね)
⑦(8) で「三角」さんに、「青」くなれとお願いし、続けて色を塗っています。
⑧中が青くなった「三角」を (9) で動かしています。(0, 200) は上に 200 の方向です。

- 1 -

3. カメ太を歩かせよう！

ドリトルでは、1 行ずつ入力して動作を確認することができます。「ペンなし」とすると移動するときに絵を描かず、「ペンあり」にすると絵を描きながら移動します。また、例 1 の (2)(3)(4) を例 2 の (10) のように記述することもできます。今回は、タートルから「カメ太」を作り、表示エリアで歩かせてみましょう。

①ドリトルを起動します。デスクトップのドリトルアイコンの上でダブルクリックします。
②次にコードを入力します。テキスト入力ラインへ、以下のコードを入力してみましょう。

```
（例 2）
カメ太=タートル！作る。
カメ太！ペンなし   90 度   左回り   200 歩   歩く。  ….    (10)
```

③さあ、実行してみましょう。[実行] ボタンをクリックして、カメ太の動きを確認してみましょう。

4. プログラムを動かしてみよう！

例 1 のプログラムのファイルを読み込んで実行し、動作を確認しましょう。

①「sample1.txt」を読み込みます。ファイル名の枠内にファイル名「sample1.txt」と入力し、[読み込み] ボタンをクリックします。
②[実行] ボタンをクリックすると、三角形が表示されます。

5. プログラムを変更してみよう！

例 1 のプログラムを変更して、大きさや角度の異なる 2 つの三角形を作成し、表示させましょう。このとき、1 つめの三角形の名前を「三角 1」、2 つ目の三角形の名前を「三角 2」とします。
また、「カメ太」を表示エリアの中央に戻すには、「カメ太！　戻る。」とします。図形の色には、「黒」、「赤」、「青」、「緑」、「黄」、「白」、「水」、「紫」が指定できます。
（ヒント：「三角 = カメ太！　図形にする。」を「三角 1 = カメ太！　図形にする。」に変更すると？）

①プログラムを編集します。テキスト入力ラインのプログラムを変更します。
②プログラムを保存します。ファイル名の枠内にファイル名を入力し、[保存] ボタンをクリックします。

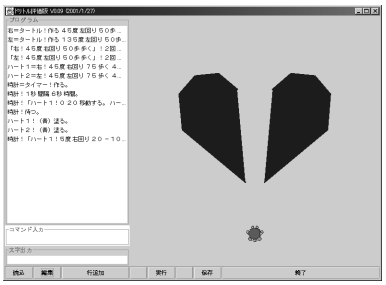
[課題 1]
①正三角形の図形を作成してみましょう。
②正六角形の図形を作成してみましょう。
③大小複数の四角形の図形を作成してみましょう。

- 2 -

図 10 テキスト例（第 1 回）
Fig. 10 Text sample.

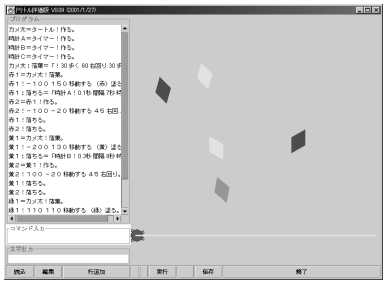
A.3 GUI の例

図 9 に配列と GUI 部品を使ったプログラムを示す．実行すると，画面にテキストボックスと「入れる」「見る」と書かれた 2 個のボタンが表示される．テキストボックスに「みかん」などの単語を入れて「入れる」ボタンを押すと,「バスケット」という名前の配列に値が入る．続けて「りんご」「バナナ」などを入力する．次に「見る」と書かれたボタンを押すと，テキス



```
右＝タートル！作る 45 度 左回り 50 歩 歩く。
左＝タートル！作る 135 度 左回り 50 歩 歩く。
「右！45 度 右回り 50 歩 歩く」！2 回 繰り返す。
「左！45 度 左回り 50 歩 歩く」！2 回 繰り返す。
ハート 1 ＝右！45 右回り 75 歩く 45 右回り 171 歩く 図形にする (赤) 塗る。
ハート 2 ＝左！45 左回り 75 歩く 45 左回り 171 歩く 図形にする (赤) 塗る。
時計＝タイマー！作る。
時計！1 秒 間隔 6 秒 時間。
時計！「ハート 1！0 20 移動する。ハート 2！0 20 移動する」実行。
時計！待つ。
ハート 1！(青) 塗る。
ハート 2！(青) 塗る。
時計！「ハート 1！5 度 右回り 20 -10 移動する。
  ハート 2！5 度 左回り -20 -10 移動する」実行。
```

図 11 高校生の作品例（1）
Fig. 11 Student's program (1).



```
カメ太＝タートル！作る。
時計 A ＝タイマー！作る。
時計 B ＝タイマー！作る。
時計 C ＝タイマー！作る。
カメ太：落葉
  ＝「！30 歩く 60 右回り 30 歩く 120 右回り 30 歩く 閉じる 図形にする」。
赤 1 ＝カメ太！落葉。
赤 1！-100 150 移動する （赤）塗る。
赤 1：落ちる＝「「 ！18 右回り 5 -5 移動する」実行」。
赤 2 ＝赤 1！作る。
赤 2！-100 -20 移動する 45 右回り。
時計 A！0.1 秒 間隔 7 秒 時間。
赤 1！落ちる。
赤 2！落ちる。
黄 1 ＝カメ太！落葉。
黄 1！-200 130 移動する （黄）塗る。
黄 1：落ちる＝「「 ！30 右回り 10 -13 移動する」実行」。
黄 2 ＝黄 1！作る。
黄 2！100 -20 移動する 45 右回り。
時計 B！0.3 秒 間隔 8 秒 時間。
黄 1！落ちる。
黄 2！落ちる。
緑 1 ＝カメ太！落葉。
緑 1！110 110 移動する （緑）塗る。
緑 1：落ちる＝「「 ！40 右回り -10 -10 移動する」実行」。
時計 C！0.2 秒 間隔 6 秒 時間。
緑 1！落ちる。
カメ太！ペンなし 90 左回り 200 歩く 90 右回り -250 歩く。
カメ太！ペンあり 500 歩く 図形にする （黄）塗る。
```

図 12 高校生の作品例（2）
Fig. 12 Student's program (2).

トボックスに配列の中身が「みかん りんご バナナ」のように表示される．

A.4 実験授業のテキスト例

図 10 に実験授業の初回で用いたテキストを示す．あらかじめ用意されたサンプルプログラムを提示して動作を確認した後，自分で打ち込んで実行，プログラムの修正，課題プログラムと進む．2 回目以降の授業でも，同様の 2 ページの資料を配布して進めた．

A.5 高校生の作品例 1

図 11 は高校での実験授業で作られた作品プログラムである．このプログラムは 2 個のタートルオブジェクトを生成し，それぞれが左右に対象に動いて赤いハートを描く．描かれたハートから左右の半分ずつの図形オブジェクトを生成する．左右の図形オブジェクトは同時に（あたかも 1 個の赤いハートのように）上昇し，突然青い色に変化して左右に分かれて落下する．

A.6 高校生の作品例 2

図 12 は高校での実験授業で作られた作品プログラムである．このプログラムは落ち葉に見立てた 3 色の平行四辺形の図形オブジェクトを生成し，それらを独立のタイマーで回転させながら落下させる．落ち葉の動きは，それぞれの図形オブジェクトにメソッドとして定義している．このプログラムでは，同じ色の 1 枚目の図形オブジェクトが 2 枚目以降のオブジェクトのプロトタイプになっており，1 枚目に定義したメソッドを共有して動作する．また，色のグループごとにタイマーを用意しているため，同色の図形オブジェクトは 1 枚ずつ順に落下する．

(平成 13 年 3 月 12 日受付)
(平成 13 年 6 月 21 日採録)

兼宗　進（正会員）

1963 年生．1988 年千葉大学工学部電子工学科卒業．1990 年筑波大学大学院理工学研究科理工学専攻修士課程修了．同年（株）リコー入社，現在に至る．2000 年から筑波大学大学院経営・政策科学研究科企業科学専攻博士課程在学中．ACM，IEEE 各会員．プログラミング言語，データベースシステム等に興味を持つ．

御手洗理英（正会員）

1960 年生．1978 年目黒星美学園高等学校卒業.（株）フジミックを経て,（株）アーマットを設立．ソフト開発に従事し，現在に至る．1985 年放送大学教養学部産業と技術専攻卒業，2001 年筑波大学経営・政策科学研究科経営システム科学専攻修士課程修了．放送大学非常勤講師．著書に「学習 BDS C・α-C」（工学図書，共著）「入門 Visual Basic6.0」（きんのくわがた社）等がある．日本ディスタンスラーニング学会，日本教育工学会，教育システム情報学会各会員．

中谷多哉子（正会員）

1980 年東京理科大学理学部応用物理学科卒業．日本電子計算（株），富士ゼロックス情報システム（株）等を経て 1995 年よりエス・ラグーン代表，現在に至る．1994 年筑波大学大学院経営・政策科学研究科経営システム科学専攻修士課程修了．1998 年東京大学大学院総合文化研究科広域システム科学系博士課程修了．博士（学術）．ソフトウェア分析方法論，要求工学．共編著に「ソフトウェアパターン」（共立出版）等．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE 各会員．

福井　眞吾（正会員）

1959 年生．1984 年東京工業大学大学院理工学研究科情報科学専攻修士課程修了．同年日本電気（株）入社．現在，同社戦略マーケティング本部エキスパート．1991 年〜1992 年イリノイ大学客員研究員．現在，筑波大学大学院経営・政策科学研究科博士課程在学中．プログラミング言語，分散システム，データベースシステム等に興味を持つ．ソフトウェア科学会，ACM 各会員．

久野　靖（正会員）

1956 年生．1984 年東京工業大学理工学研究科情報科学専攻博士後期課程単位取得退学．同年東京工業大学理学部情報科学科助手．筑波大学経営システム科学専攻講師，同助教授を経て現在同教授．プログラミング言語，プログラミング環境，ユーザインタフェース，情報教育に興味を持つ．著書に「UNIX による計算機科学入門」（丸善),「入門 WWW」（アスキー）等がある．日本ソフトウェア科学会，ACM，IEEE Computer Society 各会員．