初心者入門用言語「若葉」の言語仕様と処理系の実装

吉良智樹[†] 並木美太郎[††] 岩崎英哉[†††]

プログラミング入門用の言語が初心者にとっては習得が難しく，その役割を果たしきれていないという現状がある．そこで最初に学ぶべき言語として，必要最低限で簡潔な仕様を用意することで言語の習得を容易にし，制御構造やデータ構造，アルゴリズム概念などのプログラミングの基本を容易に学ばせるための初心者入門用言語「若葉」を設計した．主な仕様としては，初心者にはそれほど必要とされない機能を排除したこと，データを数値型と文字列型に限定したこと，制御構造に関して一つの制御には一つの記述方法という方針を採ったことなどがある．他にも，初心者が直感的に理解しやすい特徴を持つ．処理系として，Java バイトコードへのコンパイラを Java で実装した．若葉処理系及び生成コードが Java 仮想機械上での移植性を持つことは，学習環境を選ばないプログラミング言語として入門用に適しているだけではなく，現在のネットワーク時代に対応することで初心者の意欲を高めることにもつながる．

A Design and Implementation of the Programming Language "WAKABA" for Entry Courses of Programming Educations

Tomoki Kira,[†] Mitaro Namiki[††] and Hideya Iwasaki[†††]

For beginners to learn programming, current programming languages require some labors, for example, many concepts of programming models, syntax and semantics of languages which make them hesitate to learn programming. So a new language "WAKABA" is designed for entry courses of programming with the policy to simpilfy the syntax and semantics of the language. The language WAKABA has the following advantages for programming educations: (1)The policy that the only one way is prepared for one control construction makes it easy to learn programming concepts and techniques. (2)Blocks and statements may have evaluated values to know the basis and concepts of expressions, functions and scopes. (3)Basic data types, numeric, string and matrix are supported to program essential algorithm learned by beginners. The WAKABA compiler that translates WAKABA source codes to Java bytecodes, is also implemented in Java so as to be portable. The portable compiler has a correspondence to the network with Java and an effectiveness to learn programming on multi platforms.

1. はじめに

コンピュータの需要は年とともに増え続け，学術機関における実験や研究に関しても，プログラミングを必要とする分野はより一層増加した．さらには非理工学系や小中高等学校でのコンピュータ利用教育，およびプログラミング教育が重要視されつつある．

プログラミング教育には，教材となるプログラミング言語が必要となるが，初めから C 言語[1] のようなシステム開発向きのプログラミング言語を採用すると，ポインタや構造体，プリプロセッサなど高度な技術をも参照しなければならず，習得することが増えてしまうため，理解するのが困難であることも多い．最近では，C 言語や C++[2]，Java[3),4)] といった近代的な言語を教育に用いるケースも増えてきているが，一般的にこれらの言語では学ぶべきプログラミングの概念が多く複雑であり，たとえサブセット化しても完全に仕様を絞るのが難しく，説明を曖昧にせざるを得ない部分が残るため，初心者にとっては敷居の高い言語となっている．過去には，プログラミング入門講義において，教育用言語として定評のあった Modula-2 と C++を使用した場合の比較を行った結果，理解度に

† 東京農工大学大学院工学研究科
The Graduate School of Technology, Tokyo University of Agriculture and Technology
†† 東京農工大学工学部
Faculty of Technology, Tokyo University of Agriculture and Technology
††† 東京大学大学院工学系研究科
Department of Information Engineering, The University of Tokyo

差がなかったという報告[5]もあるが，C++を用いた場合には習得が難しいであろうという経験的予測から，C++を用いたクラスの評価が甘くなった可能性もあるとしており，さらにはC++のioストリームなどいくつかの概念に関しては，Modula-2での同等の機能に比べて習得が難しかったという結果も出ている．

そのためプログラミング教育には，BASICやPASCALといった言語が使用されることが多かった．対話型プログラミング環境への要求から生まれてきたBASIC[6]，COBOLやFORTRANで不十分であった構造化プログラミングを意識したPASCAL[7]，概念獲得形成の視点から設計されたLOGOなど，これらの言語は従来の言語は教育用としては向かないという判断の元で開発されてきた言語であり，教育に対して一定の効果を挙げてきたことが知られている．例えば著者らの所属する東京農工大学の電子情報工学科・情報コミュニケーション工学科においても，プログラミング序論（以下プロ序）というプログラミング入門講義において，BASICの標準規格であるFullBASICの処理系TrueBASICを用いて講義を行ってきた実績がある．しかし現実問題として，初心者入門向けのプログラミング言語を使用しても，プログラミングの技術を習得するのはなかなか難しく，その役割を果たし切れていないというのが現状である[8]．BASICについては，変数の型やスコープについての概念が曖昧であること，同様な処理に対してキーワードの羅列による多彩な記述方法があることなどが入門用言語としての問題点と考えられるが，一番の問題は言語仕様の拡張化が進み，従来の初心者向けという概念からはかけ離れた巨大なプログラミング言語となってしまい，多くのプログラミング嫌いを作り出す要因にすらなっていることである．例えば，プロ序で実施したアンケート結果[8]より，TrueBASICの言語仕様が複雑で難しいという声や，より簡潔な仕様を求める声が多く集められた．またLOGOに関しても，基本的なプログラミング概念を習得できているはずなのに，C言語などの一般的な言語にスムーズに移行できているとは言えないという報告[9]などがある．

また，Visual BASIC©やVisual C++©といった言語が初心者にはわかりやすいという理由で教育用に用いられることが多くなってきたが，Windowsという環境に強く依存していること，イベント駆動方式という特殊なプログラミング方式を採用していることを考えると，初心者教育には適していない．

このような背景から，著者らは徹底的に初心者入門用にこだわった言語があってもよいのではないかと考え，新しいプログラミング言語の設計を行った．本研究における「初心者入門用プログラミング言語」とは，これからプログラミングを勉強しようという人が最初に習得しやすく，かつ他のプログラミング言語にも応用ができるような言語を意味する．ここで「初心者」とは，必ずしも将来計算機科学や情報工学の専門に向かう人だけを対象としているわけではなく，大学の初年度に相当する人を対象として想定している．

初心者入門用のプログラミング言語に求められるのは，習得が容易であること以外に，プログラミングの煩わしさをそれほど感じさせず，初心者が興味を持てるような題材のプログラミングが簡単にできるようにすることである．そこで本研究では，簡潔な文法，簡略化された機能，Javaによるネットワーク指向，機種非依存といった特徴を持つプログラミング言語「若葉(WAKABA)」を設計し，その処理系を実装した．

2. 本研究の目的

プログラミング教育において重要なのは，プログラミング言語の文法を教えるのではなく，プログラミング概念を教育するということである．例えば，次のような事項があげられる．

(1) プログラム構造の理解

変数の扱いやサブルーチンなどの考え方を身につけ，データ構造，制御構造といったプログラミングに関する基本的な概念を理解させる．

(2) アルゴリズムの概念の理解

プログラミングとアルゴリズムの関わりについて学習をさせ，プログラミングによってどのようなことが実現できるかを理解させる．四則演算，数学の公式や定理などの実装，物理シミュレーション，文字列処理，データのソーティングや格納，検索などのデータ処理，などが考えられる．

これらの概念はプログラミング言語に依存しない分野なので，他の言語に移行したときにも応用が効き，容易にその言語を習得できる．このことは，プロ序で実施したアンケート結果[8]にも現れている．

以上を受けて，既存言語での問題点をまとめると次のようになる．

(1) 覚える事柄が多く，それに労力が割かれる

既存の言語では，構造体や分割コンパイルなどの初心者には不要と思われる機能や，複雑な文法や特徴が多く，全てを覚えるのに苦労している．著者らは大学工学部においてプログラミング基本教育を担当してきたが，そこで問題となっていたこととして，同じ制御(ループ構造など)を処理するのにいくつもの記述方法

があることや，数値データに整数型や浮動小数点型などいくつもの種類があることなどが，初心者にとっては混乱の原因となることがあげられる．例えば，「10回まわって脱出するループ」を記述することを考えた場合，従来の言語の多くではfor，whileなどいくつかの記述方法があり，どれを使えばよいのか悩んだり，他の記述方法と混在する記述を行ってしまったりなど，ループ構造という基本的な構造に対して非常に労力を必要とする結果に陥っている．さらにBASICでは，関数とサブルーチンについて明確に区別をしており，定義方法だけでなく引数の渡し方についても値渡しと参照渡しで異なり，統一性のなさが顕著に表れているために習得が難しくなっている．実際にプロ序でのアンケート結果を見ると，学生のプログラミング技術習得率はあまり高いとは言えず，もっと「単純で簡潔な書式」がよいと答えた学生が半数以上を占めている．また，既存言語の仕様を絞って教える場合でも，C言語のincludeやJavaのクラス概念，publicやstaticなどのアクセス修飾子など，省くことのできない記述を「おまじない」として曖昧な説明をせざるを得ないという問題がある．

(2) 学習環境の違いによる混乱がある

プログラミング言語の教育では，教育を受ける側が自分の手でプログラミングを行い，実際に動作させることが必要不可欠であり，教育現場における計算機環境は非常に重要な要素となる．しかし，現状としては処理系のプラットフォーム変更時の改版の遅れや費用など問題は多く，どのような計算機環境でも安定したプログラミング学習環境を整えるということは難しい．また，最近では自宅に計算機を所有する学生も多いが，自宅と学校の異なったプラットフォームや処理系のライセンスの問題などにより，作成したプログラムをそのままでは移植できないといった問題が発生している．

そこで本研究では，次の目標を定めた．

(1) 簡潔で理解の容易な言語仕様

仕様を簡潔にして言語自体の習得を容易にし，基本的なプログラミング概念を習得することを目的とする．言語仕様については3章で述べる．

(2) ネットワーク指向のプログラミング環境

コンパイラそのものと若葉プログラムをJava仮想機械(JavaVM)[3),4)]上で実行することにより，システムや若葉プログラムのポータビリティやネットワーク指向など，初心者がどこでも学習できるプログラミング環境を提供することができる．言語処理系については4章で述べる．

```
num a, b, c, d, e;     //数値変数宣言
str x;                 //文字列変数宣言
a = 10;                //10 進表示
b = 010;               //8 進表示
c = 0x10;              //16 進表示
d = 10.5;              //固定小数点表示
e = a + b + c + d;     //数値演算
x = "Wakaba";          //文字列データ
```

図1 データ型の例
Fig. 1 Example of data types.

3. 若葉の言語仕様

プログラム学習を効果的に行うためには，習得が容易になるように簡潔な仕様を持ち，プログラミング技術を習得するための必要最低限の機能さえ備えていればよい．また，直感的に初心者に理解しやすい仕様を持つことも望まれる．そこで若葉は，次に示すような方針で言語仕様を設計した．

(1) 習得しやすい簡略化された仕様

(2) プログラム概念の学習に向いた仕様

若葉の言語モデルは手続き型言語である．文の逐次実行が行われるトップダウン的な手法は，初心者にとって直感的に理解しやすいためである．また，変数の代入や参照が容易に行えること，分岐，繰り返しなどの制御構造を記述しやすいことなどからも，手続き型言語が初心者向きであると考える．

3.1 簡略化された仕様

構造体やプリプロセッサ，分割コンパイルなど，小規模なプログラミングが中心となる初心者にとっては必要のない機能を省き，ひとつの内容はひとつの機能という方針をとることで仕様を簡潔にできる．このことはプログラミング言語について習得ことを減らして学習の労力を少なくし，混乱をなくすことにつながる．

3.1.1 データ型の簡潔化

C言語やJavaなどでは豊富なデータ型を持ち，データ型の変換，ポインタ等が初心者の習得を妨げる原因となっている．若葉では扱うデータ型を次に示す二つに限定し，概念の簡潔化を行った．図1に例を示す．変数をまとめて扱うことができる配列は，プログラミングにおいて重要な要素である．そこで若葉では配列についても，数値データ用と文字列データ用を用意した．

また，値の不定な変数をなくすために，変数や配列は宣言された時点で初期値として数値型は0，文字列型は空文字列（""）を格納する．

(1) 数値データ型

全ての数値データを倍精度浮動小数点数(64ビット)に統一することで，整数のバイト数や浮動小数点数について区別する必要がなくなり，初心者が容易に数値データを扱えるようになる．また10進，8進，16進の表示を行うことができる．

(2) 文字列データ型

配列やポインタを使うことなく文字列を変数に格納して扱うことができ，初心者にも扱いやすい．

(3) 真偽値

Javaでは比較演算や論理演算の結果はブール値となり，他のデータ型への代入は許されていない．また，条件文の条件部にはブール値しか与えることができない☆．しかし若葉では数値型によってこれらを表現し，値が0のときは偽となり，0以外なら真として扱う．条件式の条件部にも，数値型の値や式を与えられるようにした．

3.1.2 一制御一構文

TrueBASICやC言語などの既存の言語では，複数種類のループなど，制御構造を用途によって使い分ける．そのため，基本的には同じ制御なのに複数の記述方法が存在する．このことは，初心者に習得することを増大させ，混乱の原因にもなる．若葉では次に示すように，一つの制御に一つのプリミティブな構文を用意することで混乱を防ぎ，習得を容易にする．また，一制御一構文の特徴を持つことによって，プリミティブな構文を組み合わせてプログラムを構築する必要があるため，言語に依存しない構造化技術の習得が可能となる．

(1) 条件分岐

if-else式だけを用意した．C言語などにあるswitch文に相当するものは，複数のif-elseにより実現する．

(2) 繰返し

無限ループだけを用意する．脱出条件をループ内の任意の場所にif式を用いて設定することで，forやwhileのような特別な文法を用意する必要がない．図2に例を示す．

(3) 脱出構文

ループ，関数などすべてに共通なブロック脱出構文を用意することで，ブロックからの脱出をすべて同じ概念として捉えることができる．マルチレベルブレークはなく，一番内側のブロックから脱出する．

(4) 関数定義

☆ しかし内部的にはブール値のtrueは整数値1，falseは整数値0に変換されており，構文規則によってその制限を与えているだけである．

```
num x; x = 0;
loop{                  //無限ループ開始
  x = x + 1;
  if(x>10)exit; //x が 10 を超えたら脱出
};
```

図2 繰返しの例
(1から10まで足すプログラム)
Fig. 2 Example of loop.

```
func num whatday(num y, num m, num d){
  num c,w1,w2,w3;       //関数内局所変数
  c = floor(y/100);
  y = floor(y-100*c);

  if(m <= 2){y = y-1; m = 12+m;};

  w1 = floor((21*c)/4);    //floor() は
  w2 = floor((5*y)/4);     //組込み関数
  w3 = floor((26*(m+1)/10));

  exit (w1+w2+w3+d-1)%7;  //値を持って関数脱出
}
```

図3 関数定義の例
(曜日算出関数)
Fig. 3 Example of definition of function.

いくつかの処理をまとめて記述する方法として関数定義がある．定義された関数はプログラム中から関数名で参照され，値を持つことができる．関数もデータ型として，数値データ型と文字列データ型を持つ．図3に例を示す．

3.1.3 要素の式化

関数型言語を除いた従来の言語の多くは，プログラム中に文と式が混在していた．つまり，プログラム中の要素には値を持つものと持たないものがある．この混乱をなくすため，若葉においてはプログラム中の要素はすべて式として扱うことで統一した．例えば，図4に示すようにブロックが値を持つことにもなる．このことにより，関数だけでなくすべての要素がブロック脱出構文によって値を持つことが可能，と概念を統一できる．この仕様は関数プログラミングの概念の土台ともなり，関数型言語の習得も容易にする．

3.1.4 関数引数の参照渡し

C言語のように変数を値渡しにすると，配列が参照渡しなので概念の統一が取れず，混乱の原因となる．すべて参照渡しにすることで，変数と配列を同じように扱うことができる．また，ポインタによって引数を

```
x = {                  //ブロックをxに代入する
  num y;               //局所変数yの宣言
  y = getnum(); //キーボードから入力
  if(y >= 0)exit y   //正ならその値
  else       exit 0; //負なら0が値となる
};
```

図4 要素の式化の例
Fig. 4 Example of conversion to expression of factor.

渡す必要もなく，複数の返り値を返すことができる．

3.2 技術習得に向いた仕様

前節で述べたように，若葉の仕様は習得が容易となるように簡略化されている．それに加えて，次に示すようにプログラミング教育に適した仕様を持つ．

3.2.1 変数定義とスコープ概念

変数や関数は使用前に宣言をすることにより，その型の種類を明らかにする必要がある．宣言はプログラムの先頭とそれぞれのブロックの先頭に記述でき，スコープはブロック内に局所的に扱われる．しかしプログラム全体は暗黙的にひとつの大きなブロックを形成すると解釈されるので，プログラム先頭で宣言された変数は大域変数としてのスコープを持つ．この仕様により，変数の宣言とスコープの概念を習得できる仕様となっている．

3.2.2 組込み関数の扱い

初心者に対して関数の概念を統一するため，組込み関数はC言語のインクルードやJavaのインポートのような記述を必要とせずに，ユーザ定義関数と同様に記述して扱うことができる．また，Javaのクラスライブラリを流用して組込み関数として追加することができるので，グラフィックやネットワークといった初心者が興味を持ちやすい内容についても容易に対処することが可能である．

3.2.3 日本語文字の採用

若葉では，識別子，文字列，コメント文については日本語の使用を許可している．日本人の初心者がプログラムを見たとき，プログラムの内容を把握しやすいためである．一方，予約語および組込み関数はすべて英字で定義されているので，日本以外で使用する際にも問題はない．

3.3 若葉のプログラミング例

前節までに述べた仕様を用いて，実際にプログラミングした例を示す．図5は，ニュートン法によって平方根を求めるプログラムである．若葉では，関数定義はプログラムの先頭で行う(1)．これは，「宣言は先頭で」の概念に従った仕様である．absの中のexit式は，関数定義を囲むブロック，すなわち関数からの脱出となる(4)．それに対して，ブロックに含まれているときのexit式は，そのブロックからの脱出命令である(10)．数値型変数はa，b，cの3種類宣言されている(5)が，さまざまなデータ型をすべて同じ型の変数として扱うことができ(6)(7)(8)，異なるデータ型の計算結果もまた，同じ数値型として扱うことができる(11)．print関数(12)は組込み関数であるが，ユーザ定義関数の使用(10)と同様の記述で使用可能である．

次に図5のプログラムを，若葉の特徴を用いて少し変更したプログラムを図6に示す．日本語識別子が使用可能であるから「平方根」という名前の変数が定義でき(5')，変数に入る結果が直感的に理解できる．繰返しの結果が求める平方根になっているから，繰返しの要素を変数「平方根」に直接代入することもできる(9')．これは，すべての要素が値を持つから可能となる記述である．しかし，ブロック脱出の際にブロック値となる値を持って脱出しなければならない(10')．

4. 若葉処理系の実装

前章まで若葉の言語仕様について述べてきた．この章では若葉の処理系の実装について述べる．

4.1 処理系の構成

若葉の処理系はJava言語によって記述することで，Javaの特徴である移植性をもった処理系となっている．つまり，異なった計算機環境下でも同じ処理系を使用してプログラミングを行うことができ，2章で述べた学習環境の違いによる混乱を防ぐことが可能である．現在実装しているのはコンパイラと実行時ライブラリであり，Javaのクラスライブラリも一部利用可能である．

処理系の実装を考えるときに，インタプリタによる実装という選択肢も考えられるが，この場合はプログラムを実行するためのインタプリタを同時に配布しなければならないという問題があり，また処理系の速度についてもやや難がある．そこで若葉は，コンパイラによって若葉のソースコードからJavaのバイトコード[10]を生成する．このバイトコードはJavaVM上で実行される．このことにより若葉処理系で生成されたコードは移植性を持ち，学校では動作したのに家では動作しないということや，教科書どおりに記述したのに動作しない，といった混乱が起こらない．また，自分が作ったプログラムをホームページ上で公開したいという場合，若葉で作られたバイトコードによる配布を行えば，実行するOSの種類や若葉のシステムの有

```
func num abs(num x){      //(1) 数値型関数の定義
  if(x<0)exit x           //(2) 条件分岐 真のとき (4) 関数脱出
  else   exit -x;         //(3)          偽のとき (4) 関数脱出
}
num a,b,c;                //(5) 数値変数の宣言
a = 2;                    //(6) 整数も数値型
b = 1.5;                  //(7) 固定小数点数も数値型
c = 1e-10;                //(8) 浮動小数点数も数値型
loop{                     //(9) 無限ループ開始
  if((abs(b*b-a)/a) < c)
    exit;                 //(10) ループ脱出
  b = ( b + a /  b) / 2; //(11) 数値計算の結果は数値型変数
};
print(b);                 //(12) 組込み関数による結果の表示
```

図 5 若葉のプログラム例 (1)
(ニュートン法による平方根の近似)
Fig. 5 Example of program of WAKABA (1).

```
func num abs(num x){      //(1) 数値型関数の定義
  if(x<0) exit  x         //(2) 条件分岐 真のとき (4) 関数脱出
  else    exit -x;        //(3)          偽のとき (4) 関数脱出
}
num a,b,c, 平方根;        //(5') 数値変数の宣言
a = 2;                    //(6) 整数も数値型
b = 1.5;                  //(7) 固定小数点数も数値型
c = 1e-10;                //(8) 浮動小数点数も数値型
平方根 = loop{            //(9') 無限ループの結果を代入
  if((abs(b*b-a) / a) < c)
    exit b;               //(10')b の値を持ってループ脱出
  b = (b+a/b)/2;          //(11') 数値計算の結果は数値型変数
};
print(平方根);            //(12) 組込み関数による結果の表示
```

図 6 若葉のプログラム例 (2)
(ニュートン法による平方根の近似)
Fig. 6 Example of program of WAKABA (2).

無に関わらず，JavaVM さえインストールされていればそのプログラムを使ってもらうことができる．このことは，初心者のプログラミング意欲を誘うことにもつながる．また，JavaVM 上ではバイトコードはインタプリタによって実行されるため，インタプリタの特徴である対話性，動的評価性などは実行段階において保証される．このことから，Java バイトコードへのコンパイラと JavaVM のインタプリタによる実行という組合せは，初心者入門に適しているといえる．

4.2 特徴的なコード生成方法

若葉コンパイラは Java のバイトコードを生成するため，基本的にコード生成方法は Java のコード生成方法 [3] に類似したものとなる．しかし若葉独自の仕様も多く，内部的に細かい操作を必要とする．ここでは若葉の特徴的な仕様について，バイトコードの生成方法を述べる．

4.2.1 要素の式化

若葉ではプログラム中の要素が値を持つが，基本

```
i={
  if(i >0)exit i+1  //(1) 真なら i+1
  else     exit i-1; //(2) 偽なら i-1
};
```

図 7 ブロックの値の例
Fig. 7 Example of value of block.

的にはスタックトップの値が要素の値となる．例えば x=x+1 という式の場合，生成コードは

```
0005: dload_0    //0 番目の変数 x をロード
0006: dconst_1   //double 値 1
0007: dadd       //x+1
0008: dstore_0   //変数 x にストア
0009: dload_0    //変数 x をロード
```

となり，最終的にストアされた x の値がこの式の値となる．しかしこの式の値は参照されないことも多く，pop によりスタックの値を破棄する必要があるなど無駄が多い．そのため，この式を y=(x=x+1) という形で y に代入する場合のように，値を必要とする場合だけロード命令を生成するように最適化する．

しかし，要素が関数を含めたブロックの場合は少し異なり，ブロック脱出構文 exit によって，ブロックの値を決める．

例えば，図 7 のようなソースコードの一部があるとき，生成されるコードは次のようになる．

```
0009: dload %0    //0 番目の変数 i をロード
000B: ldc2_w #34 <Double 0.0>
000E: dcmpl       //i と 0 の比較
000F: ifle $0016 //i<=0 なら 0016 番地へ
0012: dconst_1    //スタックに 1 (真)
0013: goto $0017 //0017 番地へ
0016: dconst_0    //スタックに 0 (偽)
0017: ifeq $0023 //偽なら 23 番地へ
001A: dload %0    //変数 i をロード
001C: ldc2_w #36 <Double 1.0>
001F: dadd        //i+1
0020: goto $002C //002C 番地に脱出
0023: dload %0    //変数 i をロード
0025: ldc2_w #38 <Double 1.0>
0028: dsub        //i-1
0029: goto $002C //002C 番地に脱出
002C: dstore %0   //変数 i にストア
```

つまり，図 7 のブロック中の条件式 (1) が満たされれば，001A 番地から 0020 番地のコードが実行されて

```
double ax[],ay[];    //引数分の一時配列
ax = new double[1]; //大きさ 1 で作る
ax[0] = x;          //引数の値を格納
ay = new double[1];
ay[0] = y + 1;
ftest(ax,ay);       //配列を引数に渡す
x = ax[0];          //元の変数に値を戻す
```

図 8 参照渡しの擬似化 (呼出し側)
Fig. 8 Example of call by reference.

スタック上に i+1 の値が積まれ，002C 番地のストア命令によって i に代入される．また，図 7 (2) の else がないときに条件部が満たされなかった場合や，exit の後ろに値が与えられていない場合など，値を特定できない場合はデフォルト値である 0 がこのブロックの値として代入される．

4.2.2 引数の参照渡し

Java では，引数の受け渡しはスタックに引数を積んでから invokestatic 命令を実行する．呼び出される関数側では，与えられた引数をその順番に従って局所変数の 0 番，1 番，というように順に参照できる．オブジェクトや配列はポインタを渡し，int 型のような基本データは値渡しになる．

若葉では，各引数を一時的な配列に格納することで，引数の参照渡しを実現する．例えば ftest(x, y+1) という関数呼び出しがあるとすると，x と y+1 の値それぞれを大きさ 1 の一時配列に格納し，関数を呼び出せば参照渡しが実現できる．次に，関数から処理が戻ってきた後でその配列の要素を元の変数に戻せばよい．同様の処理を擬似的に Java のソースコードで記述すると，図 8 のようになる．

x は変数が直接与えられているのでその内容が書き換えられる可能性があるが，y+1 の y についての参照が渡されるわけではないので，y の値が書き換えられることはない．実際に若葉コンパイラによって生成されるバイトコードは，次のようになる．

```
0006: dload %6    //6 番目に格納されている x
0008: dstore %10 //一時的な変数に退避
000A: iconst_1    //大きさ 1 の
000B: newarray <double> //一時的な配列を
000D: astore %12 //作って
000F: aload %12   //一時配列の
0011: iconst_0    //0 番の要素に
0012: dload %10  //一時的変数の値を
0014: dastore     //格納する
0015: dload %8    //8 番目に格納されている y
```

```
0017: ldc2_w #38 <Double 1.0>
001A: dadd       //y+1
001B: dstore %13 //一時的な変数に退避
001D: iconst_1   //あとはxと同様
001E: newarray <double>
0020: astore %15
0022: aload %15
0024: iconst_0
0025: dload %13
0027: dastore
0028: aload %12  //一時的配列を
002A: aload %15  //スタックに積んで
002C: invokestatic #37 //関数呼出し
002F: aload %12  //x用の一時的配列
0031: iconst_0   //0番目の要素を
0032: daload     //ロードして
0033: dstore %6  //xの場所に格納する
```

次に，呼び出される側の処理だが，次のような関数定義があるとする．

```
func num ftest(num a, num b){
  a = a + b;
  return a;
}
```

引数を配列として受け取るので，仮引数部はソースコードでは変数だが内部的に配列とする必要がある．このとき，局所変数の0番目と1番目に配列引数の参照が格納される．ここで局所変数の2番目と4番目☆に一時的に変数を用意し，参照渡しされた配列の要素を格納する．関数内で仮引数を参照するときにはこの一時変数を参照し，値の変更を行う．関数からexitによって脱出するときに，一時変数の値を引数として渡された配列の要素に格納することで，引数の参照渡しは実行される．擬似的にJavaのソースコードで記述すると，図9のようになる．若葉で生成されるコードは，次のようになる．

```
0000: aload %0   //一つ目の引数の
0002: iconst_0   //配列の要素を
0003: daload     //ロードして
0004: dstore %2 //一時変数に格納
0006: aload %1   //二つ目も同様
0008: iconst_0
0009: daload
000A: dstore %4
```

☆ num(=double)値は二つ分のスペースを使用するため，3番目には2番目の変数のデータの続きが入っている．

```
double ftest(double xa[], double xb[]){
  double a,b;
  a = xa[0];
  b = xb[0];
  a = a + b;
  ax[0] = a;
  bx[0] = b;
  return a;
}
```

図9 参照渡しの擬似化（呼ばれる側）
Fig. 9 Example of called by reference.

```
000C: dload %2   //一時的a
000E: dload %4   //一時的b
0010: dadd       //a+b
0011: dstore %2 //一時的aに格納
0013: aload %0   //仮引数配列の
0015: iconst_0   //要素に
0016: dload %2   //一時的aの値を
0018: dastore    //格納する
0019: aload %1   //あと一つも同様
001B: iconst_0
001C: dload %4
001E: dastore
001F: dload %2   //aの値を持って
0021: dreturn    //関数脱出
```

例では数値型データを扱ったが，文字列データに関しても同様に参照渡しが実現できる．

5. 評価

(1) 定性的評価

初心者入門用のプログラミング言語としての性能を評価するために，初心者が記述するプログラムの目安として，プロ序で使用された教科書[11]内の例題プログラムを参考にした．教科書内では教材であるTrueBASICで記述されているため，同様の内容で若葉の文法に書き直した．その結果，n人のインディアン(入出力，ループによるプログラム)，最大公約数(代入と繰り返し，判定)，ニュートン法(図5参照)，関数，サブルーチン定義のプログラム，簡単な統計処理(配列，データ，ファイル入出力)など，若葉ではまだ未実装であるグラフィックス処理やTrueBASIC独自のデータ処理☆☆を除いたプログラムをほぼ記述することがで

☆☆ DATA文，READ文というデータの格納と参照を行う独特な機能がある．

きた．プログラム行数は各々15行から25行程度(統計処理プログラムは53行)で計150行程度であり，初心者は大規模なプログラムを記述することはほとんどない．このことから，若葉が初心者入門用の講義での使用に十分な言語仕様であることが言える．

(2) 定量的評価

初心者のプログラミング学習においては，プログラム自体が比較的小さいこともあり，また高速な実行を必要とするプログラムを記述することも考えにくいので，システムの速度に対する要求は少ない．今回は，若葉のシステムの基本性能を評価するために，Javaシステムとの比較を行った．評価方法として，若葉コンパイラがソースコードからバイトコードを生成するのに要した時間と，そのコードを実行するのに要した時間を測定し，Javaコンパイラのコンパイル時間と生成コードの実行時間と比較した．テストプログラムとして，図5に示したニュートン法のプログラムを用いた．測定環境を表1に，測定結果を表2に示す．

若葉コンパイラはJavaコンパイラに比べ，最適化を行わないなど機能を限定しているためか，コンパイルに要する時間が大幅に短くなった．実行時間に関しては，若葉のほうが若干速度は劣るものの，ほぼ同程度の速度のものが生成されたといえる．コンパイル時間，実行時間ともに特別速いというわけではないが，初心者学習には十分な速度であると言える．

6. 議　論

プログラミング教育に関する研究は過去にいくつか行われてきているが，教育用のプログラミング言語としての研究は過去にあまり見られない．例えば，チャートを用いてアルゴリズム概念を教育する研究[12]や，構造化チャートを用いてアセンブリ言語を教育する研究[13]などがあるが，これらはチャートによって視覚的に概念を理解させるものであり，エディタでコードを記述し，そのコードを機械語に翻訳して実行するという，プログラミングに関する基本的な作業の流れの習得に対しては不完全である．また，C言語やJavaといった既存言語を，仕様を絞って教えることで入門用の言語とする場合も多いが，完全に仕様を区切って記述するのは難しく，クラス概念やプリプロセッサなどに触れることは避けられない．初心者がそれらの記述に対して疑問を持ったとき，結局それらの概念を教えなくてはならないか，または今はまだ覚えなくてよいと言ってその概念を隠蔽するしかない．そういう現状からも，若葉のように仕様の簡潔化を特徴とした入門用言語の設計は重要であると言える．

また，プログラミング教育をよりわかりやすいものとするためには，簡潔な仕様を提供するだけではなく，プログラミングの間違いを指摘する研究[14]のような初心者のデバッグ支援や，電子教科書を用いた教育[15]や問題解決から導くプログラミング教育[16]のように，教材についてもわかりやすく使いやすいものを用意する必要があるだろう．

表1 測定環境
Table 1 Measurement environment.

機種	Gateway G6-200
CPU	Petium Pro 200MHz
メモリ	32MByte
OS	Microsoft Windows 95
Java	JDK 1.1.5

表2 測定結果
Table 2 Measurement result.

	翻訳時間	実行時間
若葉	2.59秒	1.91秒
Java	5.45秒	1.92秒

7. おわりに

本論文ではプログラミング学習における初心者教育の現状を調査し，その結果を受けて設計された初心者入門用プログラミング言語“若葉”の仕様設計と，Javaを利用した若葉処理系の実装について述べた．これからは，教育課程の改変による情報科目の取組みをはじめとして，ますますプログラミング教育の重要性は高まっていくと思われる．教材として適したプログラミング言語があまりないという現状の中で，若葉はその教材となりうる特徴や性能を持ったプログラミング言語としての可能性を示した．今後は，初心者向けのGUIを含めたプログラミング環境の実装，エラーチェックの向上やデバッグ支援，若葉用の教材作成，およびシステムの性能評価が課題となる．また，学習段階にあわせた言語仕様の拡張を行うという方向性もある．言語仕様についての評価を行うためには，実際にプログラミング教育の現場で使用してもらう必要がある．そこで今後は，任意に十数人の被験者を用意し，比較する言語を用意してグループを分け，演習問題などを通じて習得度のチェックを行うというような実験で評価を行いたい．

参考文献

1) Ritchie, D.M.: The Development of the C Language, *ACM SIGPLAN Notices*, Vol. 28, No. 3, pp. 201–208 (1993).

2) Stroustrup, B.: A History of C++: 1979–1991, *ACM SIGPLAN Notices*, Vol. 28, No. 3, pp. 271–298 (1993).
3) Lindholm, T. and Yellin, F.: *The Java virtual machine specification*, Addison Wesley (1996).
4) 日本サンマイクロシステムズ株式会社マーケティング本部製品企画部: JAVA 言語環境 A White Paper, 日本サンマイクロシステムズ株式会社 (1996).
5) Hitz, M. and Hudec, M.: Modula-2 versus C++ as a first programming language—some empirical results, *ACM SIGCSE Bulletin*, Vol. 27, No. 1, pp. 317–321 (1995).
6) Kurtz, T. E.: BASIC, *ACM SIGPLAN Notices*, Vol. 13, No. 8, pp. 103–118 (1978).
7) Wirth, N.: Recollections about the Development of Pascal, *ACM SIGPLAN Notices*, Vol. 28, No. 3, pp. 333–342 (1993).
8) 吉良智樹, 並木美太郎, 岩崎英哉: 初心者入門用言語「若葉」によるプログラミング学習環境の設計と実現, 情報処理学会コンピュータと教育研究会, Vol. 99, No. 17, pp. 9–16 (1999).
9) 和田勉: LOGO を用いた「プログラミングの世界」への導入教育の経験, 情報処理学会コンピュータと教育研究会, Vol. 92, No. 77, pp. 9–18 (1992).
10) Gosling, J.: Java Intermediate Bytecodes, *ACM SIGPLAN Notices*, Vol. 30, No. 3, pp. 111–118 (1995).
11) プログラミング序論授業計画設計プロジェクトチーム: プログラミング序論 (1997). (東京農工大学工学部電子情報工学科コンピュータサイエンスコース内プログラミング序論教科書).
12) 山本義一, 辻野嘉宏, 都倉信樹: アルゴリズム教育を目的としたチャート型言語システム, 情報処理学会コンピュータと教育研究会, Vol. 93, No. 47, pp. 27–36 (1993).
13) 神村伸一: 構造化チャートを利用したアセンブリ言語教育の提案, 情報処理学会コンピュータと教育研究会, Vol. 99, No. 17, pp. 1–7 (1999).
14) 渡辺高司, 青木征男: プログラミング初心者を対象とした誤り場所指摘システムの試作, 電子情報通信学会教育工学研究会, Vol. 98, No. 259, pp. 41–48 (1998).
15) 高橋参吉, 渡邊耕平, 松永公廣: 電子教科書を用いたプログラミング教育, 電子情報通信学会教育工学研究会, Vol. 98, No. 259, pp. 22–26 (1998).
16) 和田勉: 問題解決から導くプログラミング教育方法の案, 情報処理学会コンピュータと教育研究会, Vol. 98, No. 102, pp. 57–64 (1998).

付　　録

A.1 若葉の BNF による言語仕様定義

```
<プログラム> = <変数定義>*<関数定義>*[<ブロック内部>]
<変数定義> = <型> <変数>[','<変数>]*';'
<関数定義> = 'func' <型> <識別子>
             '('[<仮引数>[','<仮引数>]*]')'
             <式>
<仮引数> = <型><識別子>
<ブロック内部> = <変数定義>*[<式>';']*
<式> = <ブロック式>
     |<条件分岐式>
     |<ループ式>
     |<代入式>
     |<脱出式>
     |<算術演算式>
<ブロック式> = '{'<ブロック内部>'}'
<条件分岐式> = 'if''('<式>')'<式>
               ['else' <式>]
<ループ式> = 'loop' <式>
<代入式> = <変数>'='<式>
<脱出式> = 'exit' [<式>]
<算術演算式> = <論理 AND 式>
               ['or' <論理 AND 式>]*
<論理 AND 式> = <条件演算式>
                ['and' <条件演算式>]*
<条件演算式> = <加法演算式>
               [<条件演算子><加法演算式>]*
<条件演算子> = '=='|'!='
               |'<'|'>'
               |'<='|'>='
<加法演算式> = <乗法演算式>
               [{'+'|'-'}<乗法演算式>]*
<乗法演算式> = <項>
               [{'*'|'/'|'%'}<項>]*
<項> = [<単項演算子>]<因子>
<単項演算子> = '+'|'-'|'not'
<因子> = <変数>
       |<関数適用式>
       |<数>
       |<文字列>
       |'('<式>')'
<変数> = <識別子>|<配列要素>
<関数適用式> = <識別子>
               '('[<式>[','<式>]*] ')'
```

```
<数> = <10 進数値定数>
      |<8 進数値定数>
      |<16 進数値定数>
      |<指数定数>
<10 進数値定数> = <10 進整数部>['.'<10 進数字>+]
<8 進数値定数> = '0'<8 進数字>+
<16 進数値定数> = '0x'<16 進数字>+
<指数定数> = <10 進数値定数>
             'e'['-']<10 進数字>+
<文字列> = '"'<任意の文字列>'"'
<配列要素> = <識別子><配列添字部>+
<配列添字部> = '['<式>']'
<識別子> = {<英字>|<日本語文字>}
           [<英字>|<10 進数字>|<日本語文字>]*
<英字> = 'A'-'Z' | 'a'-'z' | '_'
<16 進数字> = <10 進数字>
              |'A'-'F'
              |'a'-'f'
<10 進数字> = '0'-'9'
<8 進数字> = '0'-'7'
<10 進整数部> = '0'|<非ゼロ数字><10 進数字>*
<非ゼロ数字> = '1'-'9'
<任意の文字列> = [<英字>
                  |<10 進数字>
                  |<日本語文字>|…]*
<日本語文字> = 平仮名|片仮名|漢字
<型> = 'num'|'str'
<注釈> = '/*'<任意の文字列>'*/'
<行末注釈> = '//'<任意の文字列>
```

< >:非終端記号 ' ':終端記号
*:0 回以上の繰返し +:1 回以上の繰返し
[ ]:省略可能 |:または { }:選択 -:範囲

(平成 11 年 5 月 28 日受付)
(平成 11 年 10 月 13 日採録)

吉良 智樹（学生会員）

昭和 50 年生．平成 10 年東京農工大学工学部電子情報工学科卒業．同年東京農工大学大学院工学研究科電子情報工学専攻修士課程入学．初心者入門用プログラミング言語「若葉」の研究を通じ，言語設計および言語処理系に興味を持つ．

並木美太郎（正会員）

1984 年 3 月東京農工大学工学部数理情報工学科卒業．1986 年 3 月同大学工学研究科情報工学専攻修了．同年 4 月日立製作所基礎研究所入社，1988 年 4 月東京農工大学工学部数理情報工学科に助手として着任，1993 年同大学電子情報工学科助教授，1996 年 10 月から 1997 年 7 月まで North Carolina 州立大学に文部省在外研究員として勤務，現在，東京農工大学工学部情報コミュニケーション工学科助教授．15 年ほど OS/omicron 独自 OS プロジェクトの中核メンバとして，OS，言語処理系，ウィンドウシステムなどのシステムソフトウェア，並列処理，日本語情報処理の研究および計算機科学・工学の教育に従事する．近年は，組込みシステム，モーバイル計算機，ネットワーク，データベースなどにも興味を持つ．博士 (工学)．

岩崎 英哉（正会員）

1960 年生．1983 年東京大学工学部計数工学科卒業．1988 年同大学院工学系研究科情報工学専攻博士課程修了．同年東京大学工学部計数工学科助手．1993 年 4 月東京大学教育用計算機センター助教授．1996 年東京農工大学工学部電子情報工学科助教授を経て 1998 年 4 月より東京大学工学系研究科情報工学専攻助教授．工学博士．記号処理言語，関数型言語，システムソフトウェアなどの研究に従事．日本ソフトウェア科学会，ACM 各会員．