

プログラミング入門教育のためのプログラミング言語「若葉」とその処理系

吉良智樹、並木美太郎(東京農工大学 大学院工学研究科)、

岩崎英哉(東京大学 大学院工学研究科)

〒184-8588 小金井市中町 2-24-16 東京農工大学

TEL: 042-388-7139 Fax: 042-387-4603

e-mail: kira@namikilab.tuat.ac.jp, namiki@cc.tuat.ac.jp, iwasaki@ipl.t.u-tokyo.ac.jp

概要

本報告では、プログラミング入門用言語として、プログラミング言語「若葉」を提案する。簡潔で理解の容易な言語仕様を目標とし、学習者として大学および中等教育程度のプログラミング入門者を想定、より高度なプログラミング教育への連続性も考慮した。設計方針として、仕様の簡潔化と複数概念の干渉を排除するため、機能の単一化を定めて言語設計を行った。本言語の処理系「若葉」コンパイラは、プラットフォーム独立を目的として Java 言語で記述されている。また、実行形式として Java 仮想機械の命令列が生成されるので、学習者は、特定の環境に依存しないでプログラムをコンパイル・実行できる。

「若葉」により入門用のプログラム数十編を作成し記述能力の妥当性を、また処理系の評価を行い、プログラミング入門として利用できることを確認した。今後の課題は、実際に教育の現場で「若葉」を用いて言語仕様の妥当性を確認すること、教材の開発、エディタなどを含む支援系の開発である。

1. はじめに

筆者らは、CS/CE の専門課程において、プログラミングの入門コースを担当してきた。CS/CE の分野ではプログラミング能力の開発は必須であることは言うまでもないが、他工学の分野においても、実験のデータ収集と整理のためにプログラム開発を行ったり、計算機シミュレーションにより理論の妥当性と工学的予測を行うなど、計算機とそのプログラミングは工学にとって必要な不可欠な道具になっている。また、経済活動においては、データベースとインターネットが革新的な道具になりつつあり、非理工学系における計算機利用は爆発的に増加している。このような背景から文系における情報処理教育、さらには、高校での情報処理教育の強化が行なわれつつあるが、単に利用するだけではなく、計算機で何が、どこまでできるか、という観点からは、CS/CE における基本概念、特に計算機の基本構造と計算モデルの理解をプログラミングを通じて学習することが不可欠に思える。

このような目的から、プログラミングの学習者は、プログラミングの技術を得るために勉強するが、最初のプログラミング言語として、言語 C のようなシステム開発向きのプログラミング言語を通してプログラミングを学ぶと、ポインタや構造体、インクルード機能など高度な

技術を参照しなければならず、覚えることが増えてしまうため、理解するのが困難となる。

そこで、教育用プログラミング言語として、Basic[1]や Pascal[2]といった言語が開発され、利用されてきた。筆者らの所属する東京農工大学の電子情報工学科においても、プログラミング序論というプログラミング入門講義で、Full Basic[3]を入門用言語として用いている。しかし、入門向けのプログラミング言語を使用しても、CS/CE の基本概念とプログラミングの基本技術を習得には困難が伴う。実際にプログラミング序論の講義でアンケートを実施した結果、プログラミング学習に苦労したという声が多かった。様々な理由はあるが、初心者入門用のプログラミング言語として、Basic に問題があると認識している。

本報告では、入門用プログラミング言語であるプログラミング言語「若葉」の設計とその処理系について述べる。「若葉」という名前は、初心者入門用のプログラミング言語ということもあり、自動車の初心運転者用のマークである若葉マークを由来としている。

2. プログラミング教育におけるプログラミング言語と処理系の問題点

筆者らは、大学工学部の CS/CE において、プログラミングの基本教育を担当してきた。教育の主たる目的は、次の2点である。

- (1) 計算機科学・工学(以下 CS/CE)の基本概念の理解

抽象化されたプログラム記憶方式のアーキテ

The Programming Language "WAKABA" for Programming Entry Courses in Education and its Compiler
Tomoki KIRA, Mitaro NAMIKI(Tokyo University of Agriculture and Technology), Hideya IWASAKI(Tokyo University)

クチャ、その上でのアルゴリズム、データ構造、計算量などの概念を教育し、CS/CE の専門課程の基礎を学習する。

(2) プログラミングの基本技術の習得

実際に問題を理解、抽象化し、プログラムとして記述、実行して動かす能力を身につけ、CS/CE の深い理解を促す。

筆者らの所属する学科では、

- (1) 1年次後期開講「プログラミング序論」
- (2) 2年次前期開講「手続き型プログラミング」
- (3) 2年次前期開講「計算機械」
- (4) 2年次後期開講「関数型プログラミング」

の4科目において、CS/CE の基本概念とプログラミング技術の習得を教育してきた。「プログラミング序論」では、Basic を用いて、CS/CE の基本概念とプログラミングの基本技術の習得を、「手続き型プログラミング」では言語Cを通して、ポインタ、構造体などの複雑なデータ構造とより実的なアルゴリズムを学ぶ。同時期に「計算機械」により、計算機アーキテクチャの入門を世界初のプログラム記憶方式のEDSACを通じて学ぶ。これら三つの講義により、現代の主要たるプログラム記憶方式と手続き型プログラミングパラダイムを理解する。プログラム記憶方式をより抽象化の進んだ形式でとらえる関数型プログラミングパラダイムを「関数型プログラミング」で理解し、関数型によるプログラミングの形式化の理解をLispにより記述することで学び、さらに論理型などの他の抽象化の方式へつながる。上記の課程を通じて、CS/CE の基本概念、計算機の基本原則、プログラミングの方法論、プログラミング技術を養ってきた。

上記の教育は、筆者らのほか延べ20人近い教官によって担当され、一定の成果をあげてきた。5年ほど前から筆者らが担当したが、そこで発生した問題点は次のとおりである。

(1) プログラムの入門教育用言語として Basic は不適切

1年次前期「プログラミング序論」では、Kurtz と Kemeny らの設計した Full Basic 規格を用いた。現代流の制御構造、スコープを有しており、慣れたプログラマにとって記述性は高いが、次の2点が問題である。

(a) 入門者にとって記述が複雑で多様

望む制御を構成するために、for/while/repeat などの複数の方法を取りえるが、入門者の理解のためには、簡潔で解が決

まりやすい方が望ましい。ある一定の方法を理解した上で多様性を教えた方が理解しやすいと考える。

(b) 過剰なデータ構造

実世界の記述では、データ構造を構成する機能が不可欠だが、制御構造を処理の流れと比較的容易に認知できるのに対し、データ構造の抽象化は初心者には理解が困難な事項の一つである。例えば、整数型と実数型の区別と理解でさえ、最初は障壁の一つであり、機械語を学んだ後の計算機の内部構造を理解した上で初めて理解できる。実世界の情報の抽象化は、初心者にとって困難な要素の一つであり、ポインタやレコード型(構造体)は、専門課程の学生でも理解に労力を費やす者もいる。

(2) 言語教育の連続性

Basic を学んだ後に、言語CやLispを学ぶ。最初はBasicとの違いにとまどう学生も多く、Basicとの対応関係を示して理解を促したこともあった。最初から言語Cで教える、という選択肢もあるが、機能が多いことからBasicよりも初心者向けのプログラム言語に不向きである。仕様が複雑ならサブセットで記述すれば、という意見もあるが、必ず利用せざるをえない機能が残り、そこは「おまじない」として例えば、言語Cの“#include”などは最初のうちは、あいまいにせざるをえない。また、Lispを通じて関数型プログラミングを教えているが、すべての処理を関数として構造化すること、型を意識しないプログラミング、値名前・実体結合の柔軟性、すべての式が値を持つなど、プログラミングとして有用な考え方を教えているが、Basicと言語モデルと表記が大きく異なり、移行に手間がかかっている。

(3) 実習用の教育用計算機環境と処理系の問題

プログラミングの教育は、実際に受講者自らが例題プログラムを実行し、また、問題を自分で解いてプログラムを作成することが必要不可欠である。教育用計算機は、汎用大型機の集中型から分散型のワークステーションの時代を経て、現在はWindows 9X ないしはNTなどのPCを中心とした構成が主流となっている。教育の観点からは、どのような計算機構成でも、適切な機能を提供できる環境が望ましいが、実際は計算機の入れ替えごとに環境の再構築を強いられているのが現状であろう。例えば、筆者らはFull Basicの処理系の一つであるTrue Basicを用いてきたが、OSの改版への対応の遅れ、OS

の改版時の処理系のバージョンアップ費用の捻出、開発会社の消失など処理系の提供に問題があった。同様の問題は、他社製品でも存在している。さらに、近年は自宅に PC を所有している学生が増加しており、自宅での学習環境を望んでいるが、種々のプラットフォーム、処理系の価格とライセンスが問題となり、学校と同じ環境とすることが難しいことがある。

以上は、教える側が感じた問題点であるが、受講者が感じている問題点をアンケートで確認した(1997年度、52人分)。結果を図1に示す。書式が簡単でない、環境がわかりにくい、などの問題点を指摘し、より簡単な書式を望む学生が多かった。

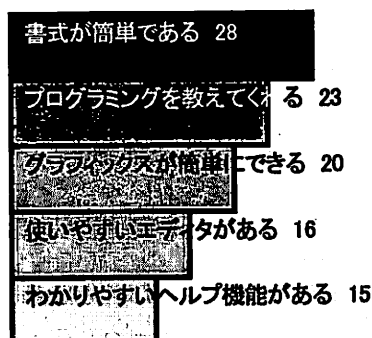


図1. 受講者が望む機能

さらに、近年、筆者らの学科は、改組を行い情報コミュニケーション学科と改名した。改組後の学科では、従来の情報専門教育だけでなく、言語文化コースにおいて、別の情報産業の担い手になりうるコンテンツ産業を支える人材育成を新たな目標として掲げている。そこで、先の4教科に加え、1年次前期において「コンピュータ序論」なる科目において計算機リテラシーを教え、使い方を学ぶ。専門課程の計算機科学コースと言語文化コースのコース分け前までは、先に述べた科目を通じて、CS/CE の基本概念とプログラミングの基礎を学ぶが、コース分け後は、計算機そのものを構築することを目標とするCSコースに比べ、言語文化コースではCSコースほど、計算機の深い理解と能力の応用は要求されないと推測される。

ここで新たな問題として、入門教育における多様な目標、があげられる。つまり、いずれは計算機の、そして、プログラミングの専門家を育てる教育とともに、CS/CE の基本概念とプログラムで何がどこまでできるかを知っていれば

十分で、プログラミングそのものに精通する必要のない人材育成、の2種の目標を達成する必要がある。

以上の議論は、筆者らの大学での問題点であるが、例えば大学の一般情報処理教育、初等中等教育の情報処理教育においてもプログラミングが必要となるが、いくつかの部分については、同様の問題点があると考えられる。

3. プログラミング入門用言語「若葉」とその処理系の目標と方針

本研究では、第2章で示した問題点の解決をはかる。簡潔な言語モデルと文法構造を有する教育用プログラミング言語を設計、処理系を開発し、プログラミング入門の教育を行う。

対象は、大学1年～2年生、可能であれば高校生を視野に定める。教育の目標は、計算機の利用者が知るべきCS/CEの基本概念とプログラムの基本技術を習得し、計算機で何ができて何ができないかを理解することから、CS/CEの専門家の入門用として機能し、他言語の習得の基盤となることまでを目指す。今後プログラミングをより深く学ぶ学生にはその基盤を提供し、そうでない学生には、簡単なプログラムにより計算機の基本概念を理解できることを目指す。

教育すべき事項として、CS/CEの基本概念とは、先に述べたようにプログラム記憶方式の抽象化としての計算モデルとその記述としてのプログラム、アルゴリズムと配列程度のデータ構造の概念、計算量などである。さらに、実際にプログラミングすることにより、プログラミングとアルゴリズムの関わりについて学習させる。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、ソーティング、検索、などを100行～数百行程度のプログラムを作成することで体得する。

プログラミング言語については、既存の言語のサブセットを用い、学習者のレベルに応じて必要な文法を追加するアプローチもあるが、第2章で述べたように、必ず説明不可能な部分が現れることから、本研究では、新規に言語仕様を定めることとした。

言語仕様とその処理系の目標を次の2点に定めた。

(1) 簡潔で理解の容易な言語仕様を有する教育用プログラミング言語の設計

初心者にとっては覚えることが少なければ少ないほどよい。必要最低限の機能に限定し、理解を容易にすることを言語設計の目標とする。

言語設計の方針として、似たような処理は単一の概念で記述することとした。例えばループ構造を記述するには、一般に for/while/repeat などの方法があること、数値データには、整数表現や浮動小数点表現に加え語長を考慮した基本型があること、ポインタやレコードなどの高度なデータ構造があることなどは、初心者にとっては混乱の原因となる。そこで、言語仕様として、処理記述の多様性を排除し、概念の単一化を行うことで、簡潔で少ない基本要素によりすべての処理を記述できる言語仕様を目指す。無論、少ないことで記述できることが限定されてはならない。基本的かつ本質的な少ない要素でも構成することで、複雑な機能を構築できることを学ばせることが CS/CE の重要な素養の一つだと考える。

(2) 移植性の高い処理系を中心としたプログラミング環境の構築

プラットフォームの多様化、自宅での教育環境への要求から、言語処理系の移植性を実装の最大の目標とする。インターネット時代を迎えて、「いつでも、どこでも、誰でも」教育・勉強できることは不可欠になりつつある。そこで、「若葉」コンパイラと「若葉」プログラムを JavaVM[4] 上で実行することにより、システムや「若葉」プログラムの移植性を確保でき、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。

4. 「若葉」の言語仕様

入門用のプログラミング学習を効果的に行うため、習得しやすい簡略化された仕様、プログラミング概念の学習に向けた仕様とする。言語モデルとしては、現在のプログラム記憶方式の根底をなす、手続き型モデルとした。文の逐次実行が行なわれ、トップダウン的に処理が記述される手法は初心者に理解しやすい。

「若葉」で記述されたニュートン法のプログラム例を図2に示す。以降の節で「若葉」の特徴的な言語仕様を示す。「若葉」の形式的な言語仕様は、付録を参照されたい。

4.1 簡潔なデータ型

言語 C や Java などの実用的なプログラミング言語では対象世界の抽象化のため、豊富なデータ型を持っている。しかし、初心者にとって、手続きの考え方は比較的理解は容易だが、データ型は、計算機の内部構造の理解とデータの抽象的な考え方が不可欠なことから、初心者の習得を妨げる原因となっている。型のない仕様も考慮したが、異なる内容、例えば文字と数値間での演算規則の複雑化をさけるため、「若葉」では基本データの型を次に示す二つに限定し、型の概念の簡潔化を行うことでその理解を容易にした。

<pre>// 平方根を求める例 num 打ち切りの値, 繰返し回数, i ; num テスト用の値[4] ; func num 絶対値を求める (num x) { if (x < 0) exit x ; else exit -x ; } func num 平方根を求める (num x, num 反復回数, num 相対誤差) { num 今の値 ; if (x < 0) exit -1 ; else if (x == 0) exit 0 ; 今の値 = x ; 反復回数 = 0 ; exit loop { 反復回数 = 反復回数 + 1 ; if ((相対誤差 = 絶対値を求める (今の値 * 今の値 - x) / x) < 打ち切りの値) exit 今の値 ; 今の値 = (今の値 + x / 今の値) / 2.0 ; } } テスト用の値[0] = 0.5 ; テスト用の値[1] = 2.0 ; テスト用の値[2] = 3.0 ; テスト用の値[3] = 9.0 ; 打ち切りの値 = 1e-6 ; i = 0 ; loop { if (i >= 4) exit ; print (平方根を求める (テスト用の値[i], 繰返し回数, 誤差)) ; i = i + 1 ; }</pre>	これはコメント (a) プログラム内の大域な数値型変数定義 (b) 数値型の配列 (c) 数値型関数定義 (d) exit で値を持って関数から脱出 (e) 引数は参照渡し (f) 関数内の局所変数 (g) 繰返したが値を持ち、しかも関数からの返り値 (h) ここで繰返しの脱出 (i) ここから主プログラムの手続き (j) 配列添字変数を伴う繰返し (k) 繰返しの終了
---	--

図2. 「若葉」によるニュートン法のプログラム例

(1) 数値型

全ての数値を倍精度浮動小数点数(64 ビット)に統一することで、整数や浮動小数について考える必要がなくなり、初心者が容易に数値データを扱える。数値型は定義 num で指定される(図 2-a/b/f, 図 3)。なお、数値データは比較演算や論理演算の真偽値としても使用される。

(2) 文字列型

文字列データの扱いは重要なので、文字列型を特別に用意した。文字列型は定義 str で指定される(図 3)。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。

```
num d, f ; // 数値型変数定義
str x ; // 文字列型変数定義
f = 99.9 ;
d = 5 * (f - 32) / 9 ;
x = "摂氏と華氏" ;
print(x) ; print(f) ; print(d) ;
```

図 3. 数値型変数と文字列型変数の例

これら二つの基本型から配列(図 2-b)を構成することができる。繰り返すが、型を構成する要素は、ポインタやレコードではなく、多次元を許す配列に限定し、理解を容易にする。本格的な型の構成は、本言語の理解習得後と考える。

変数は明示的に定義することにより、その種類を明らかにするとともに、暗黙の型や変数の利用を許さない様を行う。

4.2 一制御一構文

Full Basic や言語 C などの既存の言語では、繰返しの制御構造や多方向の選択構造などに対して、複数の記述法を用意することで、処理記述の簡潔性と明確性を示せるように設計されている。しかし、同じような制御構造なのに複数の記述方法が存在することは、初心者に見えることを増大させ、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基本的な構文を用意し、まず、プログラムが単一の基本構成要素から導出され、構成されていくことを理解させる。本質を理解しておくことで計算の概念の基本、他言語への移行は容易になると考える。また、同時に、複数の記述法による混乱を防ぎ、覚える項目を減らし、プログラミングの概念と技術の習得を容易とし

た。

(1) 関数とブロックによるプログラムの構成

手続き型のプログラミング言語では、処理単位の抽象化の方法としてサブルーチンの考え方が重要となる。既存の言語では、値を返すものをサブルーチン、値を返さないものをサブルーチンと呼び、異なる形式で定義を行う言語が多い。「若葉」では、値を返さないサブルーチンも関数に統一する。関数定義は、func で始まり数値型か文字列型を指定する。なお、主プログラムについては、関数の外で記述された処理を主プログラムとしている(図 2-i)。

また、関数内においてもさらに詳細な処理単位が存在するが、「若葉」は“()”で囲まれた範囲をブロックとして定義し、処理単位を示す。なお、後述するが、「若葉」ではブロックも値を持ち、処理を Lisp 的に記述できる(図 2-g)。

(2) 条件分岐

if-else 式(図 2-c/g)だけを用意した。言語 C などの switch 文に相当する多方向の選択構造は、if-else の連なりにより実現する。

(3) 繰返し

単に繰返すだけの構文 loop だけを設定した(図 2-g/j)。脱出条件については、繰返し内の任意の場所に if-else 式を用いて設定し、for/while/repeat 型だけでなく、1/2 繰返しも基本要素から構成できる。

(4) 脱出構文

繰返し、条件分岐、関数などすべてに共通なブロックの脱出構文 exit を用意することで、ブロックからの脱出を同一の概念としてとらえる(図 2-h/k)。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。また、exit に値を書くことで関数を含むすべてのブロックの評価値を返すことができる仕様とした(図 2-g/h)。

4.3 参照渡しによる引数

サブルーチンや関数においては、引数の扱いが問題となる。言語 C や Java 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とれず、混乱の原因となる。Pascal では、仮引数定義で、var を前置することで参照渡しを指定していた。Full Basic では、サブルーチンは参照渡し、関数は値渡しという仕様で紛らわしい。「若葉」では、すべて参照渡

しとし、仕様の一貫性を確保した(図 2-e)。

4.4 要素の式化

「若葉」において、プログラム中の要素は関数定義を除き、すべて値を持つとして扱う。例えば、図 2-g 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。

4.5 その他の仕様

(1) 変数・関数定義とスコープの概念

プログラミングの基本概念としてスコープの理解は重要である。「若葉」では、Pascal のような入れ子構造、言語 C のような分割コンパイルは不要と考え、単一ファイル、ブロックを基本としたスコープを有する設計とした。スコープはブロック内に局所的に定義される。関数も名前のついたブロックである。また、主プログラムは、“{}” のないブロックとして解釈し、プログラム先頭で宣言された変数は、ファイル内の大域変数としてスコープを持つ。複雑すぎない範囲でスコープの基本概念の学習を可能とした。

(2) 日本語の文字列と識別子

「若葉」では、文字列リテラル、コメントに日本語が書けるだけでなく、識別子にも日本語を許す仕様とした。識別子までを日本語とすることで、各変数や関数が何をするか、すぐに理解できる。

以上に述べた機能以外に、入出力に対する考察が重要であるが、入出力については、第 5 章で述べるようにグラフィックス機能の提供を目指しており、設計は未完成である。現在、文字ベースの入出力機能を組み込み関数として提供しているにとどまっている。

5. 「若葉」学習システムの設計と処理系の実装

5.1 全体構成

本研究では、「若葉」の学習に対して、統合化された学習システムを提供する。学習システムの最大の目標は、プラットフォームに依存しない、インターネット時代に対応した環境を提供することにある。特に、移植性の高い学習環境として、Java によりシステムを記述することで、特定機種に依存しない環境を構築できるほか、ネットワークを通じた処理系の配布、対話的な教師と学習者のコミュニケーションを行える可能性がある。また、種々の学習環境の全体構成を図 4 に示す。いずれの機能も HTML ないしは Java 言語で記述し、WWW ブラウザでの参照や JavaVM で動作することを目標としている。次の機能を想定している。

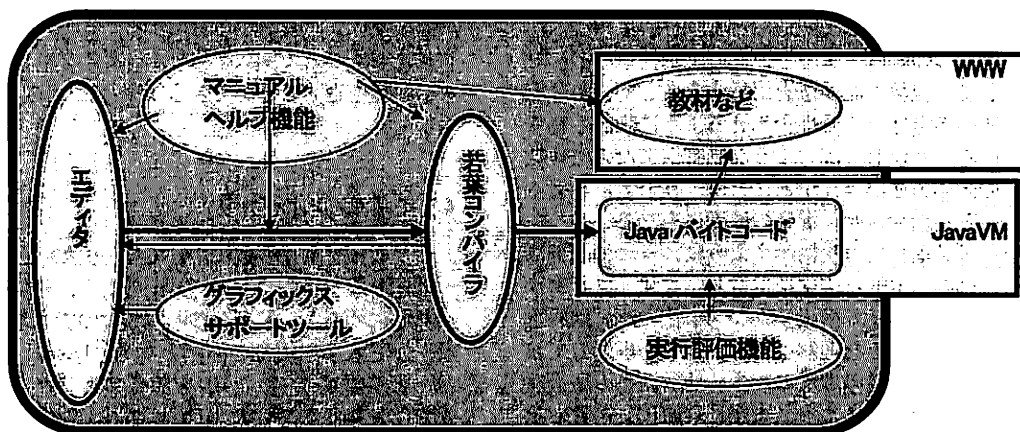


図 4. 「若葉」学習システムの全体構成

(1) 「若葉」コンパイラ

「若葉」ソースプログラムを JavaVM コードへ翻訳する。詳細は後述する。

(2) テキストエディタ

「若葉」のソースプログラムの編集を行う。WWW ブラウザのアプレットとして動作する。

(3) マニュアル・ヘルプ機能

(4) 教材

HTML など記述することで、ネットワークでの配布、利用を可能とする。現在、教育の現場でもインターネットを用いた学習を行うことも珍しくはない。そこで、ネットワーク上に教科書があれば、教科書を容易に手にいれることができるうえに、教科書の改訂もリアルタイムで行うことができ、プログラミング学習の共有もできる。

(5) グラフィックサポートツール

初心者にとって、グラフィックスによる描画は、興味をひくプラスの要因となる。Logo の成功もタートルグラフィックスによるところが大きい。簡単な描画ツールにより、グラフィックスとプログラミングの対応を容易にし、興味を持てる環境を提供する。

(6) 実行評価機能

プログラムのテスト、デバッグでは、実行時の挙動を追う必要がある。また、プログラムの評価に対する考え方を身につけさせることは重要となる。そこで、「若葉」プログラムの挙動を解析するツールを提供する。

上記の各機能のうち、現在、中心となる「若葉」コンパイラ的设计と実装が進んでいる。次節で「若葉」コンパイラについて述べる。

5.2 「若葉」コンパイラの実装

初版の「若葉」コンパイラの実装の目的は、コンパイル速度や実行速度などの性能よりも、まず、「若葉」の言語設計の妥当性を実際に検証できる環境を構築することにある。現在の実装では、教育用のエラーメッセージ生成を考慮していないが、次の版でわかりやすいエラーメッセージを考察する予定である。

現在まで実装されている「若葉」コンパイラの構成を図5に示す。「若葉」コンパイラは、「若葉」ソースプログラムを JavaVM コードに変換する。「若葉」プログラムは、「若葉」クラスライブラリとともに、JavaVM 上で動作する。また、「若葉」コンパイラを Java により記述することで、「若葉」コンパイラも JavaVM で動作

する。

現在、付録で示された言語仕様のコンパイラが、アプレット中で動作している。Java で約 3000 行程度の大きさとなっている。入出力やグラフィックスなど、今後詳細な設計を進めたときに、組み込み関数および構文から、Java クラスライブラリを組み合わせることで容易にコードを生成できるようにするため、表による定義と wrap のための機能を処理系に組み込んである。

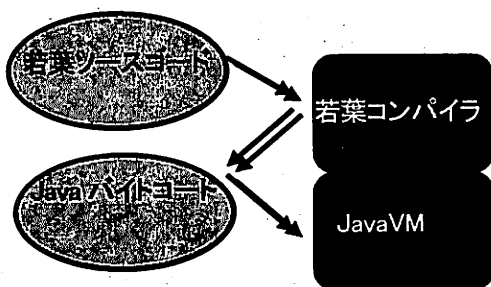


図5. 「若葉」コンパイラの構成

6. 評価

(1) 記述性の評価

現在の「若葉」の言語設計について、プロ序の教科書内のプログラムを「若葉」に書き直すことで記述性を確認した。具体的な内容としては、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、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、グラフィックス処理や Full BASIC 独自のデータ処理を除いたプログラムを記述することができた。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行程度である。また、プロ序以外の、線形検索、二分検索、ハッシュ法などの基本的な検索アルゴリズム、種々の $O(n^2)$ のソートアルゴリズム、クイックソートなどのソートアルゴリズムなども記述できている。

これらのプログラムを作成できたことで、「若葉」が初心者入門用の講義で現れるプログラムを記述できる能力があることを確認した。

これらのプログラムが初心者にとって他の言語よりもわかりやすいか、また、初心者が「若

葉」で記述しやすいかなどの評価は今後の課題となっている。

(2) 処理系の性能

本実装の目的では、コンパイラのコンパイル速度や「若葉」プログラムのオブジェクト性能は本質的ではない。しかし、極度に遅いと学習に支障をきたすため、先のプログラミング序論のプログラムのいくつかに対して、コンパイル速度と実行速度を実測した。比較のために、同等のプログラムを Java で記述したが、ほぼ同等の性能が得られており、学習に支障をきたさない性能であることを示すことができた。

7. おわりに

本報告では、プログラミング学習における初心者の現状と要求を調査し、その結果を参考にした初心者入門用に適したプログラミング言語「若葉」の仕様設計と、「若葉」を使用した学習に効果的なプログラミング環境の設計について述べた。プログラミング環境についての実装、及びその性能評価が今後の課題となる。性能評価に際しては、実際にプログラミング教育の現場で使用してもらうことを考えている。

参考文献

- [1] Thomas E. Kurtz: BASIC, ACM SIGPLAN NOTES, Vol. 13, No. 8, pp. 103-118, 1978.
- [2] N. Wirth: Recollections about the Development of Pascal, ACM SIGPLAN NOTES, Vol. 28, No. 3, pp. 333-342, 1993.
- [3] John G. Kemeny & Thomas E. Kurtz (株式会社フェムテック 訳): TrueBASIC リファレンスマニュアル, 啓学出版, 1991.
- [4] Tim Lindholm & Frank Yellin: The Java Virtual Machine Specification, Addison-Wesley, 1996.
- [5] 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 監訳: 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.
- [6] James Gosling: Java Intermediate Bytecodes, ACM SIGPLAN NOTES, Vol. 30, No. 3, pp. 111-118, 1995.
- [7] 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, 東京農工大学工学部電子情報工学科, 1997.

付録 若葉の BNF による定義

```

<プログラム> = <静的変数定義>*
               <関数定義>*
               [<メイン部プログラム>]
<静的変数定義> = <変数定義>
<関数定義> = func <型> <識別子>
              ([<仮引数定義>[, <仮引数定義>]*]) <式>
<仮引数定義> = <型> <識別子>
<ブロック式> = [<ブロック内部>]
<メイン部プログラム> = <ブロック内部>
<ブロック内部> = <動的変数定義>* [<式>;]*
<動的変数定義> = <変数定義>
<変数定義> = <型> <識別子>[, <識別子>]*;
<型> = num | str
<式> = <ブロック式>
      | <条件分岐式>
      | <ループ式>
      | <代入式>
      | (<式>)
      | <算術演算式>
      | <脱出式>
<条件分岐式> = if (<条件式>) <式> [else <式>]
<ループ式> = loop <式>
<代入式> = <識別子> = <式>
<条件式> = <式>
<脱出式> = exit [<式>]
<算術演算式> = <単項演算式> | <論理 OR 式>
<単項演算式> = [<式>]
<論理 OR 式> = <論理 AND 式> [| <論理 AND 式>]*
<論理 AND 式> = <条件演算式> [& <条件演算式>]*
<条件演算式> = <二項演算式>
               | [<条件演算式> <二項演算式>]*
<二項演算式> = <項> [[+|-]<項>]*
<項> = [+|-] <要素> [[*|/]<要素>]*
<要素> = <識別子>
        | <配列>
        | <関数適用式>
        | <数>
        | <式>
<条件演算式> = == | != | < | > | <= | >=
<識別子> = [<英字> | <日本語文字>]
          | [<英字> | <10進数字> | <日本語文字>]*
<関数適用式> = <識別子> ([<引数>[, <引数>]*])
<配列> = <識別子> <配列添字部>+
<配列添字部> = [<式>]
<数> = <10進数字>+ |
      0<8進数字>+ | 0x<16進数字>+
<10進数字> = 0-9
<8進数字> = 0-7
<16進数字> = <10進数字> | A-F | a-f
<英字> = A-Z | a-z |
<コメント> = //<任意の文字列>*
            /*<任意の文字列>*/
<文字列> = "<任意の文字列>"
<任意の文字列> =
              [<英字> | <10進数字> | <日本語文字> | ...]*

```