

# 複数のプログラミング言語を学ぶ意義

まつもと ゆきひろ



皆さんは「サビア・ウォーフ仮説」をご存じでしょうか。これは言語学における古典的な仮説の一つで、「人間の思考は使用する言語とそれに付随する文化に影響を受ける」というものです。もし仮に数字を3までしか持たない言語があったとすると、その言語を使用する文化に生まれ育った人間は3以上の数を認識できない、といったことです。言語学的にはこの仮説は否定されているようですが、日常生活の中では、この仮説が本当ではないかと感じる経験がたびたびあります。

例えば、私は年に数回海外に出張して講演をする機会があります\*1。海外の人のほとんどは日本語を理解しないので、英語で話すことになります。私も日本人の一人として当然のように英語に対する苦手意識がありますが、理解してもらえない日本語よりは、つたない英語でも1024倍くらいマシだと思って、体当たりで英語に挑戦して

います。

そこで奇妙なことに気づきました。普段、日本語を話しているときと、英語で話しているときとで、自分の性格というか態度が微妙に違うのです。日常会話（とあらかじめ準備した講演）くらいならようやくなんとかなる程度の英語力なのですが、英語で話しているときのほうが、もう少し論理的で大胆な話し方をするように思います。

プログラミング言語も、同じように人間の思考に影響を与えます。私がBASICでプログラミングに入門したばかりの中学生のころ、Pascalを学ぼうと教科書を一生懸命読んだのですが、再帰の部分がどうしても理解できず苦労したことを覚えています\*2。再帰という概念が存在しないBASICによって思考が狭められていたとしかいいようがありません。同様のことは他のプログラミング言語でも発生します。例えば「本物のプログラマはどんな言語の上でもFORTRANプログラムを書ける」というFORTRANを皮肉った有名な文章があります\*3。私自身、どう見てもCOBOLにしか見えないCプログラムを読んだことがあります。

このように、自分が普段から使っている言語が自分の思考を狭める働きをすることはあまり知られていない事実です。逆にPascalが私に再帰を教えてくれたように、新しい言語を学ぶことによって新しい概念を身に付けることができます。プログラミング言語はプログラミングの際に用いる重要なツールの一つですが、プログラマの思考に対して良いほうにも悪いほうにも影響を与

えるということを、ときには思い出す必要があります\*4。

## 毎年一つの言語を学ぶ

名著「達人プログラマー システム開発の職人から名匠への道」(ピアソン・エデュケーション発行)の中で、David Thomas氏とAndrew Hunt氏は次のように書いています。「毎年少なくとも一つの言語を学習する

言語が異なると、同じ問題でも違った解決方法が採用されます。つまり、いくつかの異なるアプローチを学習することにより、幅広い思考ができるようになるわけです。この提案に従い、その年に学ぶプログラミング言語を「Language of the Year」、略してLotYと呼びます。

多くのプログラマにとって、見知らぬ新しい言語を学ぶことは、おっくうで面倒なことです。新しい言語を学びたくないという理由で、従来のツールに固執することも珍しくありません。しかし、新しい世界が広がるのであれば、新たなプログラミング言語を学ぶことはそんなに悪いことではありません。考えてみれば、プログラミングの基礎になる部分はどの言語でもある程度共通です。新しいプログラミング言語の概要を理解するには数日あれば十分です。言語を学ぶことの最大の障害は、「勉強しても使わないから」ということかもしれません。しかし、LotYの考え方では学ぶことそのものが目的で、学んだことを日常的に使うかどうかには固執しません。

「達人プログラマー」の二人の著者は最



まつもと ゆきひろ  
(Yukihiko "Matz" Matsumoto)

ネットワーク応用通信  
研究所 特別研究員  
鳥根の田舎に住みながら

国際的なオープンソース・ソフトウェアの開発に挑むプログラマ。家族6人で幸せな田舎暮らしを満喫している。バグと原稿の締め切りがなければもっと幸せなのに、と思いつつ、考えてみれば、それらが無いなら、別の困ったことがあるよなあと思う今日このごろ。



初のLotYとしてRubyを選び、それが気に入ったため「プログラミングRuby」(第1版はピアソン・エデュケーション発行、第2版はオーム社発行)を書いたというエピソードがあります。数年前、Dave Thomas氏に会ったときに「Rubyの次のLotYに何を選んだのか」と聞いてみたところ「Haskellを試そうとした。でも、次に考えてるのは日本語だよ」と笑顔で答えてくれました。Dave Thomas氏にとっても毎年一つというのはチャレンジのようです。

## どの言語を学ぶべきか

では、LotYを実践するとして、どのような言語を学ぶべきでしょうか。前提条件としては

- ・ 仕事で使わない
- ・ 普段あまり使わない
- ・ 学び甲斐のある概念がある

ことだと思います。仕事で使う言語なら、別にわざわざLotYに選ばなくても学ぶ動機があります。「仕事に使わなくても自分の世界を広げてくれる」のがLotYのいいところですから。同様の理由から、仕事以外で頻繁に使う言語も対象外です。JavaやPHPなどはLotYの対象にはなりにくいでしょう。

LotYとして学ぶ以上、今まで知らなかった文法、機能、文化などを持つものが望ましいでしょう。世の中にはたくさんの言語がありますが、これらの条件を満たしそうなものをいくつか紹介しましょう。

### Lisp

Lispは今でも生き残っている数少ない「いにしへの言語」です。同じ1950年代生まれのメジャーな言語は、もうFORTRANとCOBOLしか残っていません。しかし、Lisp

はただ古いだけではありません。Lispの持っている特徴は以下のようなものです。

#### ・ 最古の動的言語

RubyやPythonをはじめとする動的言語が注目されていますが、Lispは最古の動的言語です。動的言語で注目されている特徴のほとんどをはじめから備えています。

#### ・ 最高レベルのリフレクション

プログラム自身を操作するメタプログラミング機能の元祖はLispです。この機能をリフレクションと呼びます。Lispは強力なりフレクション機能を備えています。

#### ・ 言語自身の拡張機能

Lispにはマクロと呼ばれる機能があり、制御構造や文法にいたるまで拡張できます。

Lisp学習の難点として、歴史的に外の世界とのインターフェースが不十分だった点があります。長い歴史があるLispにはしっかりとした規格があるのですが、各種OSでの移植性を考慮した結果、WindowsやUNIXといった特定のOSに依存する機能は使いにくくなっています。しかし、最近では入手しやすく、またOSの機能を直接利用できる処理系が増えてきました。私のお勧めは、Lispの方言の一つであるSchemeの処理系、Gauche (<http://practical-scheme.net/gauche/>) です\*5。

### Haskell

次に紹介するのは関数型言語です。C言語でも関数という単語を使いますが、関数型言語と呼ぶときの「関数」は数学的な関数です。関数型言語では数学をベースとした計算モデルを採用しています。LispもLambda計算というモデルが基礎になってはいますが、手続き型の記述もできるので、

意外に普通の言語として使えます(カッコだらけの見かけ以外は)。しかし、関数型言語の代表ともいえるHaskellは、基本的に副作用\*6がなかったり、必要なときになってはじめて計算が行われる「遅延評価」という機能が基本になっています\*7。「普通の言語」とはずいぶん雰囲気異なります。これこそ「世界を広げる言語」ではないでしょうか。Haskellをはじめとする関数型言語は近年あちこちで注目されています。また、darcのような分散バージョン管理システムやPerl 6の処理系であるPugsがHaskellで実装されたりと、実用的なプログラムが記述できることも証明されつつあります。

もう少し「普通の言語」に近い関数型言語としてはOCamlがあります\*8。OCamlは副作用があったり、遅延評価が標準でなかったりと、Haskellほどとんがっていませんが、その分、学びやすくなっています。

LotYの対象になりそうな言語はまだまだいくらかあります。例えば、ほぼすべてのオブジェクト指向言語の先祖と呼んでもよいSmalltalkが挙げられます\*9。私が開発しているRubyは、Lispっぽい動的な機能(の一部)やSmalltalkっぽいオブジェクト指向機能(の一部)を手軽に経験できるスクリプト言語として人気上昇中です。低レベル・プログラミング\*10が再び注目を集めていることから、C言語やアセンブラに回帰してみるのも意外に面白いかもしれません。

このような言語についてはこれからもいろいろいろい形で紹介していきたいと思います。現在、「Rubyist Magazine」(<http://jp.rubyist.net/magazine/>)という無料のWeb雑誌に「他言語探訪」という連載を書いているので、見てみてください。

\*1 この原稿もRuby Conference 2006のための渡米中に執筆した。

\*2 当時Pascal処理系が入手できなかったため、教科書だけで学習した。

\*3 <http://www.genpaku.org/realprogrammerj.html>

\*4 もっとも、言語以外のツールも多少ともプログラマの思考に影響を与えている。例えば、ある種のプログラマはEmacsの中毒になっている。私もそうだ。

\*5 編集部注：Gauche(ゴージュと読む)は主としてUNIXでの使用を前提にしているため、付録CD-ROM

にはScheme処理系としてWindowsで利用できるDrSchemeを収録しています。

\*6 副作用とは、処理を実行したときに「結果の値が返る」こと以外に起こる現象を指す。例えば、入力処理は副作用を伴う。

\*7 編集部注：本誌は2006年6月号特集「Haskellによる関数型プログラミング入門」でHaskellを取り上げています。また、Webで「本物のプログラマはHaskellを使う」(<http://ipro.nikkeibp.co.jp/article/COLUMN/20060915/248215/>)を連載しています。付録CD-ROM

にはHaskell処理系であるGHCを収録しました。

\*8 編集部注：Web連載「数理科学的バグ撲滅方法論のすすめ」(<http://ipro.nikkeibp.co.jp/article/COLUMN/20060915/248230/>)ではOCamlを使用しています。付録CD-ROMにはOCaml処理系を収録しました。

\*9 編集部注：本号から新連載「青木淳のSmalltalk道場」が始まりました。

\*10 低レベル・プログラミングとは「レベルの低いプログラムを作ること」ではなく、コンピュータのハードウェアに近いレベルのプログラミングのこと。