

はじめに

今までに、星の数ほどの、という大げさだが、幾百ものプログラミング言語が生まれ、そして消えていった。ここでは、現存するプログラミング言語のうち代表的なもの、消えていった言語の中でも著名なものを紹介する。ただ、資料および執筆期間の不足から、いくつかの言語は省かざるを得なかったことをご了承いただきたい。

多くの人にとっては、ここに挙げられている言語のうち、知っているものの数のほうが少ないだろう。文章を面白く書くような技能は持っていないので、退屈な文章になってしまっているかもしれないが、気楽に読んでほしい。

今回、まず最初に、古典的言語とでもいべきものを取り上げる。FORTRAN から始まった高水準言語であり、現代で使われている著名な言語のベースとなったものである。これらはデータとそれに対する処理（手続き）を分離し、処理（手続き）を主体として考えることから、手続き型言語と呼ばれる。これらの言語では処理の流れを把握しやすくするために、プログラムの構造化という考え方が提唱された。時代が移るにつれ、これらのプログラミング言語は使われなくなりつつあるが、これらの考えは受け継がれている。

次に、現在の主流であるオブジェクト指向言語。ソフトウェアは「一度完成したら変更しないもの」ではなく、「状況に応じて柔軟に改変できるもの」であることを求められた。また、ソフトウェアの規模の増大は、プログラムの構造化ではすでに収拾がつかないほどになっていた。このような問題を解決するために様々な議論が交わされて生まれたひとつの解決策、それがオブジェクト指向パラダイムである。その主なキーワードは「カプセル化」と「再利用」であり、プログラマがお互いの仕事の成果物を必要最小限の情報で利用できるようにすることである。現代のプログラミング言語は、そのほとんどがこの中に含まれている。そのうちのいくつかは手続き型言語を拡張したものであるため、ベースとなる考え方さえ理解できれば、古典的言語からの移行もそれほど難しくない。たいていは豊富なコンポーネント、すなわち再利用可能なソフトウェア部品を含んでおり、古典的言語に比較して、その生産性は一般的に高いと言える。これからは（他に新しい方法論が発明されるまでは）、このような言語が主流となるだろう。

オブジェクト指向以外にも、異なった考え方に基づいたプログラミング言語が存在している。

そのひとつが関数型言語である。これは元々、人工知能の研究などから発生してきており、関数は計算結果以外の影響（副作用）を残さない、という考え方を基本としている（例外としてファイル入出力などがある）。また、当時、それまでの言語になかったいくつかの要素を提案し、その成果は他の多くの言語にも取り込まれている。現在、特定の研究分野以外で使われることはまずないが、その思想には価値があるので、取り上げる。

他にも、スクリプト言語と呼ばれるものもある。これらは、アプリケーションの機能を補完するために、簡単なロジックを記述して実行できるようにした簡易言語のことである。例としては、UNIX系OSならシェルスクリプト、DOSならバッチファイルなどが挙げられる。たいていはその目的に応じて機能が特化されており、書いたコード量に対して処理が非常に強力であることが特徴である。手軽に書いて使える便利なツールなので、使えたと何かと便利である。

上記分類にあてはめるのが不適切だと思われた言語は、最後にその他の言語としてまとめてある。これらは独特のアイデアに基づいており、一般的にはほとんど使うことはないが、それぞれ興味深い事例なので取り上げることにした。

< 古典的言語 >

【FORTRAN】

1954年に汎用高水準言語として開発された、最古の高級言語である。当時、まだマクロアセンブラしかなかった時代に、人間の言葉に近い形でプログラムを記述でき、しかも特定のコンピュータに依存しないこの言語は、きわめて画期的なものだった。コンピュータの性能を最大限引き出せるように、つまりは無駄なく機械語命令に置き換えられるように作られており、そのあたりに、かすかにアセンブラの匂いが残っている。瞬く間に科学技術計算用として定着し、いまだに現役。おそらく、世界で最も使われている言語であろう。かつては不足がちだった記述能力も、仕様改訂ごとに向上してきている。工学系の人には嫌でも使うことになるだろうが、それ以外の人には、あまり縁のない言語である。

【COBOL】

科学計算のFORTRANに対抗できるだけの実績を持つ、古参の言語。事務処理用にはFORTRANがあまりに不向きだったので、このような言語が開発されることになったらしい。汎用言語が出回っている現在ではそれほどの有用性はなくなっているのだが、いま

だに COBOL を使っている所は多い。

【ALGOL】

COBOL と同時期に開発された言語。1960 年に発表された仕様を ALGOL60、1968 年のものを ALGOL68 という。この言語は関数の概念、BNF 記法による横文記述など、多くの点で SIMULA、PASCAL、PL/I、BCPL などの多くの言語に影響を残すことになる。しかし、仕様は次第に難解さを増していき、ALGOL68 はその膨大な仕様のため、使われぬまま消えていった。

【PL/I】

IBM 社が開発した言語。COBOL と FORTRAN、ALGOL の影響を受け、すべてのプログラミング言語を置き換えることを狙って作られた。やはり膨大な言語仕様のため、とても使いやすいとは言えないはずなのだが、まだメインフレームマシンと共に使われているらしい。

【APL】

1962 年頃に、数学的なアルゴリズムの枕念を記述するために開発された、会話型のインタープリタ型のプログラミング言語。多次元の配列の換作能力に優れ、特殊記号を多用したその独特のプログラムコードは、書くのがやさしいため、経営事務計算などにも使われたという話である。ただ、その書かれたコードを読むのは非常に困難だともいわれている。

【BCPL】

ALGOL の影響を受けて作られた言語であり、C 言語の遠い祖先にあたる。現在使われているという話は聞いたことがないし、資料も入手できなかった。

【B】

BCPL の後絶として開発された言語であり、C 言語の祖先である。現在使われているという話は聞いたことがないし、資料も入手できなかった。

【ADA】

1970年代の終わりに、アメリカ国防総省の支援のもと、軍事システム用に開発された。Pascal や Modula - 2、Algol をベースとして、言語工学の産物を積極的に取り入れた結果、処理系やハードウェアに依存しないデータの表現と操作、並行処理やリアルタイム処理、例外処理の記述能力、テンプレート機能などをサポートしたが、デバッグが難しく、危険な言語であるという指摘もあつた。

【SIMULA67】

ALGOL を基礎とした言語のひとつで、その名のとおり、シミュレーション用に関係された。現実世界をコンピュータ上に投影するために、物体に関するデータとそれに対して可能な操作の情報をまとめて扱うため、クラスという概念を実現した。この発想が、Smalltalk 社に受け継がれることになる。

【BASIC】

Beginner ' s All-purpose Symbolic Instruction Code、初心者向けの汎用言語である。古いパソコンユーザーには最も馴染み深いであろう。元々は大型計算機のタイムシェアリングシステムをインタラクティブに制御するために開発されたのだが、昔のパソコンに採用され、様々な亜種が派生していった。習得の容易さから、多くの人がプログラミングに挑戦し、パソコン雑誌にも多くのプログラムリストが投稿・掲載されていたものである。現在、正統な BASIC はほとんど使われていないが、Microsoft 氏が発売した Visual Basic が、Windows の代表的な統合開発環境として生き残っている。Visual Basic については、オブジェクト指向言語の項目で触れる。

【C】

AT & T のベル研究所で開発された高級言語。ポインタという、メモリを直接操作する手段を持っているため、ハードウェアに密着したところまで記述することが可能。UNIX をこれで記述したことがきっかけとなって、脚光を浴びる。「プログラマは賢いものである」という前提があるようで、余計なことをしないぶん高速であるのだが、他の言語での「暗黙の前提」が通用しないことがあるので注意が必要な言語である。先日、新

しい標準規格として、C#が制定された。この後継として、現在 C++と Objective C があるが、これらについてはオブジェクト指向言語の項目で触れる。この言語は現在、ほとんどすべての OS でサポートされており、特に UNIX 系の OS を使う人には必須の教養といったところだろう。

【PASCAL】

N.Wirth が 1971 年頃開発した、教育用言語。簡潔な文法ながら高度にアルゴリズム記述能力を持ち、処理系も比較的簡単に作成できるので、情報科学の基礎学習として使われることが多い。特にアルゴリズム解説の本でよく登場する。しかし、この言語にはいくつかの欠点もあり、現在では教科書以外で目にすることは少ない。

いくつかの後継となる言語があり、その中では MODULA が特に有名だろう。また、Borland 社が発売した Turbo Pascal という製品では有用な機能拡張が施されており、標準 PASCAL よりもそちらを目にする機会があるかもしれない。

現在、その後継の製品である Borland Delphi が、Object Pascal という言語をサポートしている。

Object Pascal については、オブジェクト指向言語の項目で触れる。

【MODULA - 2】

PASCAL の作者である N.Wirth は、PASCAL の次に作った MODULA という言語を作り、それをさらに改良してこの言語を作成した。PASCAL をベースとしてさらに構造化を推進し、ソフトウェアを「モジュール」という単位で複数のファイルに分割できるようになった。また、ハードウェアに密着した処理を記述するための低水準機能を含めることで、システム記述言語としての機能も上昇させている。これが後に Oberon というオブジェクト指向言語に進化するのだが、Oberon に関しては適切な資料が見つからなかったため割愛する。

【FOETH】

1969 年にバージニア州の国立電波天文台で開発された。スタック換作を基本として、逆ポーランド記法を使って記述する。逆ポーランド記法だと括弧がいらないし、式解析が簡易化できるので処理系がコンパクトになっているようである。自分で新しいコマンドを定義して、機能を自由に追加できる。日本ではマイナーだが、アメリカでは、根強い人気

があるらしい。

< オブジェクト指向言語 >

【Smalltalk】

オブジェクト指向パラダイムの基礎を築いたと言われる言語。オブジェクト空間上に存在する、一つ一つは小さな機能を持ったオブジェクトたちが互いに通信し合い、協調動作して最終的の果たすという発想である。すべてのデータはオブジェクトであるという一貫性を持っている。

オブジェクト指向について語られるときには、必ずと言っていいほど登場する言語である。この言語自体はそれほど使われていないが、その考え方には触れておく価値がある。

【Visual Basic】

Microsoft による Windows 用の BASIC 統合開発環境。この言語では、従来の BASIC にデータの構造化、モジュールやクラスの横念を導入し、カプセル化、多態性などを実現している。次期バージョンではさらにクラスの継承などもサポートするらしく、BASIC の枠を越えた言語へと進化している。修得が容易なことから、Windows ではもっとも普及する開発環境である。

商用ソフトウェアであるが、学生ならばアカデミック・パッケージで比較的安価に（それでもフリーな環境に較べれば高価だが）入手できる。

【Objective C】

C 言語のオブジェクト指向拡張で、完全な上位互換であることを目標に開発された。主に MacOS、NEXTSTEP など使われている。オブジェクト指向方法論について紹介している書待でも登場することがある。

【C++】

Smalltalk などの影響を受けて C 言語を拡張した言語。C 言語の後継として、多くのソフトウェアがこれによって記述されるようになってきた。文法はほとんどそのままに、テンプレートをはじめとする各種新機能を追加している。仕様はかなり複雑になっているが、C とまったく同じように書くことも可能なので、C 言語から無理なく移行できる。

UNIX 系 OS では GNU のコンパイラ、Windows では統合開発環境として、Microsoft の Visual C++、Borland の C++ Builder がある。

【JAVA】

Sun Microsystems Inc. が開発した、オブジェクト指向言語。開発途上では Oak と呼ばれていた。機種依存性を JavaVM という実行環境（インタープリタ）が吸収してしまうため、コンパイラが生成するバイトコードを各 JavaVM でそのまま実行できる。どんな JavaVM でも同じ動作をすることを保証するために、JAVA は C++ をベースに、その仕様の曖昧な部分（式の評価順序など）を厳密に定義している。またプログラム自身の情報にアクセスできるリフレクション機能、メモリ管理を自動化するガーベジコレクションなどを持っている。

組み込み機器の制御などでも利用しようという動きがあるし、Jini など JAVA をベースにした新技術がいくつも研究されている。多くの企業がこの言語に注目しており、将来性は期待できるはずなのだが、昨年末にも Sun は開発に絡んでいくつかのトラブルを起こしており、その先行きに不安を抱かせた。

開発環境は Sun から無料で提供されているし、他にもいくつかの統合開発環境が商用ソフトウェアとして提供されている。

【Object PASCAL】

Borland 社(のちに Inprise と改名し、その後 Corel 社と合併した)が開発した Windows の統合開発環境 Delphi がサポートする言語である。PASCAL をベースに有用な機能拡張を施し、ただの教育用言語から実用指向の言語へと変化を遂げている。C++ ほど強力ではないが、シンプルな文法のおかげで（タイプ量がやや多くなるが）ソースコードが読みやすい。

商用ソフトウェアだが、データベースなどへ接続するための便利なコンポーネントが多数付属しており、Windows アプリケーションの開発では多くの企業に利用されている。現在、Linux への移植作業が進行中とのことである。

【Eiffel】

1986 年に Interactive Software Engineering 社で開発された、純粋なオブジェクト指向言語。

”契約”という考え方に重点を置き、オブジェクトは自身の機能について事前条件、事後条件、不変条件などといった取り決めを表明できるようになっている。これによって、誤った暗黙的了解を取り除き、プログラムが安全に動作しているかを検査する機構を持つ。多重継承、強い静的型付けなどの特徴を持つ。言語仕様ではないが、Eiffel のほとんどのコンパイラは、C や JAVA など他の言語のプログラムソースを生成する。この派生で Sather という言語があるらしい。

< 関数型言語 >

【LISP】

関数型言語。LIST Processor の略で、リスト処理を主眼とした言語。起源は COBOL などと同時期で、人工知能の研究によく使われるようになった。その後多くの方言に分岐したが、その後 Common Lisp に統合された。やたらと大量に括弧が出るのが特徴で、LISP は Lots of Irritating Superfluous Parentheses (鬱陶しい大量の余計な括弧) の略だと言う人もいる。多くのソフトウェア工学の産物を積極的に取り入れる風潮があるのか、歴史が古いわりに時代遅れの言語にはなっていない。この派生として、Emacs エディタを記述するために作られた Emacs-Lisp がある。

【ML】

関数型言語。エディンバラ大学で、窟理証明支援系を記述するために開発された。シンプルな関数定義を積み重ねて強力なプログラムを記述できるという特徴があり、またファイル入出力などを含む強力なライブラリを持つ。手続き型言語をある軽度理解しているなら、視野を広げるためにこのような言語に触れてみるのもいいだろう。複雑なソフトウェアの記述用に作られただけあってか、モジュール化、カプセル化などの考え方をうまく取り入れている。

処理系として、ベル研究所の Standard ML of New Jersey などがある。

<スクリプト言語>

【AWK】

テキスト処理用インタープリタ型言語。開発者である A.V.Aho、P.J.Weinberger、B.W.Kernighan の三人の名前の頭文字をとって命名された。処理するデータを一彦の単位（普通は行）で読み込み、それに対してスクリプトに記述された処理を行う、という動作をする。最近では Perl などを使う人が多いのであまり目立たないが、即席で書くのには便利である。

【Perl】

Practical Extract and Report Language。Larry Wall が開発したスクリプト言語。awk や sed の持っていた強力な文字列操作能力と、C に近い碑文記法を持ち、CGI のようにテキストを扱う分野で大活躍している。・「やり方はいろいろある (There's more than one way to do it .)」というコンセプトで、同じ目的を達成するにも、C ライクにアルゴリズムを記述する、高度な正規表現を組み合わせる、怪しげな呪文を唱えるなど、色々な手段が選べる。強力な言語だが、オブジェクト指向に関しては後から付け足したために中途半端にしか見えず、そのあたりを嫌って Python などを使う人もいる。

【Python】

Python は 1989 年に開発されたスクリプト言語。オブジェクト指向の採用、C/C ++ 言語との組合せが容易、豊富な拡張モジュールなどの特徴を持つ。この愛好者は Perl を嫌う傾向にあると聞いたのだが、本当なのだろうか？

【Ruby】

最近一部で人気のオブジェクト指向スクリプト言語。Smalltalk などの影響を受けた、動的な型付けの言語である。Perl が 6 月の誕生石なので、Perl に続くという意味で 7 月の誕生石である Ruby から命名された。「手軽にオブジェクト指向」というコンセプトで作られており、演算子なども Perl や C に似せて直感的に使えるように図られている。is_a 関係を表す継承と、機能の共有のための Mix-in という二つの機能を持ち、多重継承よりも関係を簡潔に表せるなど、他のオブジェクト指向言語に負けない記述力を持つ。ま

だ生まれてから 5 年ほどの若い言語だが、少しずつではあるが雑誌などにもその名が上るようになってきており、将来を期待させる。

【Tcl/Tk】

The Command Language の略で、ソフトウェアのマクロ言語を統一することを目標に開発された。Tk は「Tcl 用のグラフィックライブラリで、簡単に GUI を作成できる。Tk の人気は高く、Perl を始めとする各言語でインターフェイスが実装され、利用されている。ただ、最近は Gtk (GIMP Tool Kit) が使われるようになってきたので、目立たなくなった。

< その他の言語 >

【CASL】

この言語は、第二種情報処理技術者試験の試験用に作られた。架空の計算機 COMET の上で動くアセンブラ言語で、この処理系はエミュレータ、シミュレータとして提供されている。勉強しても、資格を取る以外に使い道はないが、アセンブラの考え方を知る助けにはなるかもしれない。

【PROLOG】

論理型言語と呼ばれる言語。事象の論理的な関係や記号処理を簡単に記述できる。1971 年にフランスで開発された言語で、1981 年に日本の ICOT (新世代コンピュータ技術開発機構) が第 5 世代コンピュータプロジェクトの核言語として採用したことをきっかけに、人工知能などの各分野で使われるようになった。

【RPG】

1960 年に IBM によって開発された非手続き型言語。パラメータ形式で指示し、報告書の作成に適している。

【Mind】

日本語でプログラムを記述する異色の言語。おそらく「ぴゅう太」以来であろう。だいたい文法構造は PASCAL などに近い印象を受けた。日本語はキータイプ回数が多いのでタイピング練習にはなりそうである。UNIX 用の処理系が GPL で配布されているらしい。

最後に

プログラミング言語は、ソフトウェア工学の進歩を反映して、次々と生まれては消えていく。プログラミングを志したばかりの人にとっては、ひとつの言語を修得することが当面の目標であるのだろうし、それはきわめて大事なことである。しかし、プログラミングの勉強と、プログラミング言語の勉強は違うということを忘れないでほしい。言語はプログラミングという行為のために使う道具であり、道具の使い方をいくら覚えても、優れたプログラムが書けるようになるわけではないのだ。また、道具は次第に古くなっていくし、今使っている言語が将来も使い続けられる保証はない。ひとつの言語に固執せず、どんな言語を提示されても冷静にそれを分析するだけの柔軟な思考と広い視野を持ち、言語名を聞いただけで頭から「その言語は知らないから使えない」と決め付けるようなことがないようにするべきである。

Oh! UCC 2000 年 4 月号 ; 石尾 隆@OUCC