言語教育モデルによるプログラミング教育ツールの設計

青木 賢太郎(大同工業大学)、小川 清(名古屋市工業研究所)、廣瀬 哲也(大同工業大学)

はじめに

コンピュータには、CPUを直接制御する数値がある。CPUはこの数値（プログラム）に基づいて動作する。この数値を16進数などで表現したものが機械語である。しかし16進数では、人間がプログラムを組むのは難しい。そのため人間が扱いやすい言語を作って来たものがプログラミング言語である。

人間が扱うために作られた言語の原始的なものにアセンブラがある。アセンブラは、機械語と一対一対応可能で、数値を、命令語とレジスタ、アドレスなどの機能に分類し、文字に置き換えた物である。しかしアセンブラを実際に扱うには、CPUの機能を詳しく把握する必要がある。

1　プログラミング言語教育

ソフトウェア教育と言語

かつて、コンピュータのソフトウエアを作成するためにはプログラミング言語を覚えなければならないとされていた。

ＯＳの開発にはアセンブラ、事務用ソフトにはＣＯＢＯＬ、科学技術計算にはＦＯＲＴＲＡＮ、コンパイラの作成にはＣ、コンピュータサイエンスにはＬＩＳＰといった具合にである。

しかし、多くの場合、ワープロソフトと表計算ソフトでほとんどの用事が済んでしまう。文書処理、計算処理の概念の教育の方が重要であり、それがプログラミング教育の中心でもある。

ベンダのプログラミング言語教育

コンピュータメーカ、ソフトウェアベンダなどのプログラミング言語教育は、自社製品への囲い込み活動である。そのため、ソフトウェアの操作性、固有な文法などを覚えさせ、他の環境への移行を困難にすることが目的である。そのため、ソフトウェア開発に必要となる基本的な知識、プログラミング言語の本質などについては、捨象される場合が多い。

大学でのプログラミング言語教育

大学でのプログラミング言語教育には、教員がベンダなどのプログラミング言語教育の過程をコピーしたようなものを用いる場合があった。また、言語の文法を教えることが必要であると思いこみ、真っ白な頭に、シンタックスチェッカが処理可能な知識を詰め込もうとすることもよくあった。こういった教育方法は、最初のプログラミング言語の教育としては好ましくない。

ソフトウェアは、コンピュータを使う目的に対して開発されるべきであり、文法から何かが生まれるわけではないからである。

市工研のプログラミング言語教育

名古屋市工業研究所では、中小ソフトハウス、中小企業の情報部門向けのプログラミング言語教育を１０年以上行ってきている。その基本的なコンセプトは、問題解決であり、文法、操作方法などは、主に自習・相互学習によって修得することとしている。業務でソフトウェアを開発する場合には、定型的な操作を覚えても、それだけではほとんど役に立たないだけでなく、創造的なソフトウェア開発が行えないためである。

プログラミングの変換可能性

しかし、これらの言語の間の変換が定義されておらず、何か別の目的のプログラムを組む度に、それらの言語を覚える必要があった。それは規格においても、相互変換を規定しておらず、現在のプログラミング言語教育の重要な問題の一つである。つまり、一つのプログラミング言語を覚えれば、外の言語へは、自動変換ツールが存在

していてよいはずであるが、そのようなツールが市場性などから普及していない。

また、任意のプログラミングを組むことができる共通言語としてのプログラミング言語の定義は、他の言語からの移植が容易に行えるツールまで作られていないことから、実現していない。ただし、これらのツールの困難性は、言語の定義の変換の難しさではなく、非公開のライブラリの移植の難しさから来る場合がある。

2　プログラムの生産性

プログラムの再利用

プログラミングの生産性を向上させるには、同一のプログラムを二度書かないようにすることにある。つまり、プログラムの再利用が、プログラムの生産性向上の方法の一つである。このプログラムをライブラリと呼ぶことがある。

再利用されるプログラムは、生産性の高いプログラマによって書かれている必要がある。もし、再利用されるプログラムの質が低ければ、それ以上のプログラムを書くことができないからである。

そのため、再利用されるプログラムの生産者は、プログラムを再利用するプログラマよりも生産性が高い必要がある。

プログラミング言語の構造化

プログラミング言語のライブラリを実現する方法は、２つの方法がある。そのプログラムを直接呼び込む方法と、そのプログラムへジャンプする方法である。

前者は、データの構造化に用いることができ、後者は手続きの構造化に用いることができる。

一つの命令で多くの仕事をこなし、開発者の負担を減らしている。アセンブラであっても、データと手続きの構造化により、一つの命令で多くの仕事をこなし、開発者の負担を減らすことができる。

手続きとデータを構造化する方法があり手続きの構造化に力を入れた言語としてＬＩＳＰがあり、データの構造化に力を入れた言語としてＣＯＢＯＬがある。

関数

関数は引数と戻り値の受け渡しを自動化し、手続きを構造化するものである。つまり、関数は、アセンブラにおけるpush、popの管理、レジスタ変数の管理などを行い、手続きの構造化を容易にするものである。

初心者向け言語としてのBASIC

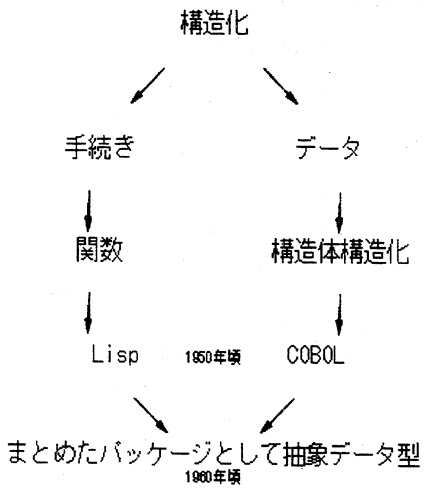
初心者向けに設計されたBASICは、関数をインタプリタで簡単に作ることができるものである。入力すると直ぐに応答がある。何かエラーがあれば、その箇所で止まって修正を受け付けるため、比較的容易にプログラムを開発できる。

しかし、BASICプログラムで表計算ソフトウェアと同等の処理を行うには、膨大な量のプログラムと、柔軟性が要求される。そのため、コンピュータの簡単な計算は、BASICではなく表計算ソフトウェアで行うことが通常となった。

しかし、表計算ソフトウェアに実装されていた自動化のためのマクロ言語が、BASICよりも古典的なアセンブラのような言語であったため、場合によってはBASICの方が使いやすい場合もあった。表計算ソフトウェアのマクロ言語がBASICになり、表計算ソフトウェアと、そのキーボードマクロによる自動化という意味でのBASICプログラムと、その修正という３段階のレベルで融合化されている。

RICSのアセンブラとしてのＣ

RISCでは、CPUの設計とほぼ同時にＣ言語が設計される場合がある。これはアセンブラがCPUごとに異なるニーモニックを用いており、異なるCPUごとに異なるアセンブラを学習するのでは、システムの開発に遅れをもたらすため、比較的実装が容易なＣ言語を、最適化コンパイラ技術により、アセンブラとCPUの機能を理解するより、容易に高速なコードを生成できるためである。ここでは、複数のプログラミング言語に共通の中間言語が開発される場合もあり、複数言語間の共通部分が明確になってきている。ただし、これらの中間言語はコンパイラ設計者のみに公開されている場合もあり、エンドユーザには気が付かない場合もある。



抽象データ型

抽象データ型は、手続きとデータをひとまとめにしたパッケージにより、手続きを含む型として、任意の型を作ることができる。この抽象データ型の技術をもとに、オブジェクト指向を実現しようとする傾向が存在した。

オブジェクト指向

オブジェクト指向は、プログラミングを機械語をはじめとする機械指向ではなく、現実に存在するものを直接操作することにより、コンピュータの専門家以外がプログラムを開発できるようにすることを考えていた。

そして、現在までに多くのオブジェクト指向言語が生まれてきた。

現実世界をシミュレートするものであるため、機械と現実との対応付けを開発者が苦労することはないと考えられてきた。

言語の比較

次に主なオブジェクト指向型プログラミング言語とその親戚筋を中心に、実装可能な言語の比較を行った。これらの評価は、具体的な処理系によって大きく異なる可能性があるため、下記の６種類の処理系が存在する機器の中で、最も多くの台数が出荷されているIBM PC互換機上の現存する処理系を元に検討を加えた。そのため、言語そのものの性格と、処理系の性格とが必ずしも評価の上で分離できていない場合もある。

簡易性は、システム起動から、実際にプログラミングするための操作全体の簡易性を評価した。

オブジェクト指向は、抽象データ型のサポートの水準で評価した。

公開性は、Cコンパイラ自体がCで書かれていること、yacc、lexなどのCコンパイラの作成ツールが存在していること、GNU Cなどソースコードが公開されていることなどから評価した。

セキュリティは、ハードウェアに対するプログラミングの可能性などから評価した。

インタプリタであるかどうかは、処理系の問題であるが、標準的な処理系がどちらを前提に設計されているかで評価した。

ダイナミック特性は、ガーベージコレクションのサポート、メモリ管理の方法などで評価した。

移植性は、ソースの公開度、現実にどれくらい移植されているかなどで評価した。

中立性は、

スレッドは、スレッドの実装などについて検討したが、基準としたOSがスレッドの実装が十分ではなかったため、十分評価できていない。

例外処理、パフォーマンスは、標準的な処理系の実装により評価した。

JAVAの課題

JAVAは、抽象データ型の言語としては、３つの問題がある。第一に、アプリケーションにおいてmain関数が存在しており、他の関数と違う仕様となっている。第二に、int型などの抽象データ型ではない型が残存している。第三に、メッセージの送受信機構に統一されておらず、様々なレベルのメッセージ送受信機能が存在していることである。

JAVAにおいても、C++と同様、抽象データ型の位置づけがわかりにくいが、インタネットでのブラウザの対応、共通のGUIなどの利点もあるため、教育対象の言語として、JAVAを想定して考察した。

言語の比較

	Java	Smalltalk	Perl	Shell	C	C++
簡易性	+	+	*	*	+	|
オブジェクト指向	+	*	|	|	|	+
公開性	+	+	+	|	*	+
セキュリティ	+	+	+	+	|	|
インタプリタ	*	*	*	*	|	|
ダイナミック特性	*	*	|	+	|	|
移植性	*	+	+	+	*	+
中立性	+	+	+	+	*	|
スレッド	+	|	|	|	|	|
例外処理	*	*	|	|	|	+
パフォーマンス	中	低	低	低	高	高

*:あり +:ある程度あり |:なし

3．人間指向

オブジェクト指向が物中心的で、静的な構造主義を前提としていたのに対して、近代的な人間主義に基づいたプログラミングパラダイムのモデルを検討する。それを人間指向と捉え、言語教育においては、母国語モデルと外国語モデルに分類する。

エージェント指向は、コンピュータの中に、人間のようなエージェントを設定して、ユーザとしての人間が、コンピュータの中のエージェントと会話をしていくことによって問題を解決しようとするものである。

あるいは、自己自身をコンピュータの中に自己のエージェントとしてシミュレートするという意味で、物中心ではないオブジェクト指向の拡張として考えることができる。 エージェントの実現には、自分自身をメッセージとして送受信することができる機構を、抽象データ型の中に持つことにより、ネットワークの中を、渡り歩きながら、問題を解決していくことを可能にする。

母国語モデル

幼児期の母国語のようにプログラミング言語を覚える方法について検討する。母国語のようにプログラミング言語を覚えるためには、母親のようにコンピュータと対話する必要がある。

そのため、コンピュータは母親のようかどうかを検証した。自分がして欲しいことをしてくれるか。それは、自分が表現する前から、自分のして欲しいことが用意されていることを前提とする。つまり、メニューが用意されている必要がある。

実際のハードウェアの面では、コンピュータの場合は、電源を入れないと動かないため、Macintoshのように電源を入れる手段と電源を切る手段が対照的ではないシステムの方が好ましい。

単語ごとに反応があるのが好ましいか、どうかの判断は、一つの動作ごとに反応するものが好ましいと仮定した。コンパイラのように、一語ごとに対する反応と、全体に対する反応と別々にあると、画面が煩雑でわかりにくいことが想定されるからである。

外国語モデル

文法中心の教育を外国語教育と命名することを考慮した。プログラムを記述することが目的の場合には、外国語教育モデルにおいても、文法中心の教育は好ましくない。

教育ツールの設計

プログラミング言語の教育ツールには、ユーザインタフェース言語を考える。従来のプログラミング言語の延長線上にある言語の改良と、ユーザインタフェースそのものの見直しによって設計するものの２つを考え、JAVAで試作中である。

参考文献

(1) ソフトウェア工学,ブルックス,岩波書店

(2) OBJの試用経験、小川清、二木厚吉、ソフトウェアシンポジウム

(3) オブジェクト指向入門、BERTRAND MEYER、アスキー出版