コンピュータと教育 40－8
（1996. 5. 24）

高校における教科「情報」としてのプログラミング教育

大岩 元(ohiwa@sfc.keio.ac.jp)
慶應義塾大学 環境情報学部

概 要

情報化社会の到来とともに、従来のリテラシー（識字）教育に代ってコンピュータ・リテラシー教育が必要となる。日本の高校における教科として「情報」を設けることを前提として、その中核となるべきプログラミング教育の理念と内容について、具体的な提案を行った。構造化プログラミングを、問題解決の方法論として用い、問題の定式化、設計、実現、評価の過程を全体として教育内容に取込み、日本語の重要性と表現技法との関連について述べる。

Programming as a Content of Informatics Education for High School

Hajime OHIWA(ohiwa@sfc.keio.ac.jp)
Department of Environmental Information
Keio University

Abstract

Computer literacy has become important part of secondary education in place of conventional literacy education because of proliferation of computers in our society. Programming education is discussed as a content of informatics for high-school students. Structured programming is presented in connection with the process of definition, design, realization(coding), evaluation of the problem to be solved. Importance of Japanese language is shown for this whole process.

1．はじめに

情報化社会の到来とともに、コンピュータ・リテラシー教育が従来のリテラシー（識字）教育に代るものとして認識されるようになってきた。文字の読み書きが出来ることが工業化社会の構成員に対して要求されると同じ意味で、情報化社会においてはコンピュータを使うことが、その構成員の基礎能力として要求される。

こうした事情をふまえてＵＮＥＳＣＯは、情報教育を充実させることを重要な社会的課題として各国に対して要望している[1]。実際、欧米の先進国は現在では、情報教育を中等教育で義務教育として行っている。中進国であるシンガポール、香港、韓国では理系に進学する高校生には情報教育を行っている。

これに対して日本では、中学・高校にコンピュータは配備されたが、情報教育は中学校では「技術・家庭」の中の選択教材として、高校では「数学」と「物理」の選択教材として行なわれているに過ぎない。中学はともかく高校では、大学に進学する学生は入学試験に出題されないコンピュータを勉強することはなく、就職する学生だけが選択している。

日本における情報教育のもう一つの問題は、教師がコンピュータ・リテリシーを持っていないことである。教員に対するコンピュータ講習会が行なわれたり、教員養成課程において、情報関係の科目が設定されているが、大部分は非常勤講師がそれぞれの信ずるところを講ずるだけであって、コンピュータ・リテラシーの育成はほとんど達成されていない。

こうした状況のもとで、日本の情報教育を世界の標準に達する所まで持ち上げるには、教科として「情報」科を設けるのがよい。本報告では、こうした意味での「情報」科における教育の中心を成す、プログラミング教育がどうあるべきかについて論じる。

2．情報教育の中核としてのプログラミング

コンピュータ・リテリシーとは何かということには議論があるが、ここでは次のように考えて議論を進めることにしたい。即ち、情報教育の目的は

1）コンピュータが利用できること
2）コンピュータの可能性と限界について理解を持つこと

である。この内、第1のコンピュータ利用のみが、日本ではコンピュータ・リテラシーの内容として考えられている。特に教育界においては、教育目的にコンピュータを利用することが強調されてきた。

しかし、コンピュータの技術進歩は速く、特定の応用プログラムの使用方法を覚えただけでは、その知識は時間とともに急速に陳腐化してしまう。このような使い方の知識は、学校教育の内容としては適切でない。コンピュータの使用を通じて、普遍的な能力が身に付くものでなければならない。これにはコンピュータに関する一般的な理解が得

られる教育を行なわなければならない。このような教育がうまく行なわれれば、技術が進歩しても、コンピュータの使用法はマニュアルを参照することにより、自習できるようになるはずである。

2．1 最良の教材としてのプログラミング

単なる使い方を越えたコンピュータの本質的な理解を持たせるには、プログラミング教育を行なうのがよい。プログラムを書く経験を持つことによって、コンピュータの動作原理が理解でき、コンピュータは何が出来て、何が出来ないかについて、直感的に理解できるからである。

プログラムはコンピュータにやらせたい仕事を正確に記述したものである。記述できない仕事を、コンピュータはやってくれない。この事を理解することは、大変重要である。例えば、人間は人の顔を見分けることが出来るが、それをどのようにして行っているかを記述することは出来ない。従って、人の顔を見分けることをコンピュータにやらせることは、少くとも人間がやるのと同じようには出来ない。

仕事の手順を正確に記述することを目的とするプログラミング教育は、単なる技術教育として以上の教育価値がある。やりたい仕事を実現するには、どのような準備が必要であるか、どんな事をどんな順番で実行していくか、後始末としてどんな事をしなければならないか、といったことを考えることは、人間の知的活動の中核を成すことであるが、従来の教育では系統的に教育する方法がなかった。

2．2 表現教育としてのプログラミング教育

コンピュータのプログラムを書くには、プログラミング言語を用いなければならない。そこで、使用するプログラミング言語の文法を理解する必要が生じる。従来のプログラミング教育はこの部分に大部分の力がそそがれてきた。文法を理解しなければプログラムは書けないが、それを理解することが目的ではない。

日本ではコンピュータ科学の研究と教育が大変に遅れている。従って、学校に限らず、プログラミング教育はプログラミング言語の文法教育になっている場合が大部分である。文法教育なら、教師にコンピュータにやらせたい仕事を正確に記述するプログラミング能力が十分になくても、教育を実施できるからである。こうした能力不足の教師が行った教育は、うまく機能するはずがない。この結果、プログラミング教育はやってもうまく行かない、という認識が教育界に広まってしまった。

こうした事情は教育界にとどまらず、ソフトウェア業界も同じである。コンピュータ科学を学んだことのない大学卒業生を採用し、社員教育でプログラムの文法教育を行って、後はオンザジョブ・トレーニングで済ませるというのが、一般的である。ここで行なわれる教育で、良いプログラムを作る方法が教育される場合は少ない。

2．3 問題の定式化、設計、実現、評価

良いプログラムを書く教育は容易ではない。何が良いプログラムかは簡単には決まらないからである。良いプログラムを書く公式は無く、経験のつみ重せが大きくものを言う。あるプログラムを、良いプログラムとも悪いプログラムとも評価することが可能で

ある。これは、何に価値を置くかによって、正反対の評価が出来るからである。

プログラムの価値は、問題をどのように捉らえるかということに依存する。従って、表現教育としてのプログラミング教育は、問題の定式化から出発しなければならない。そして、問題を設定した後、どのように実現するかの設計を行ない、続いて実現に入る。ここで初めて、文法の知識が必要となる。しかし、必要となるのは細かい知識ではなく、当面やりたい事が実現できる程度でよい。

2．4 言語能力と論理能力

問題の定式化から設計にかけては、自然言語を用いた分析作業が中心となる。どんな問題を解こうとしているかは文章で与えられるが、数学の問題と違って、一義的に解釈できるものではない。これを解釈して、一義的に解釈できる文章で表現して問題規定文とすることは、良い訓練となる。

さらに、問題が定まった所で、これを解決するための手順を考える。ここでは、言語能力とともに論理能力も必要となってくる。これを行っている間に、最初に定めた問題規定文の解釈に問題が生じる場合も多い。こうして、問題の再定式化が行なわれ、精密化されるのである。

設計が行なわれた後、それをプログラミング言語を用いて実現する。初心者の場合、文法の間違いを多くするので、これを除くことに相当のエネルギーを費やすことになる。文法エラーが無くなった所で、書かれたプログラムをコンピュータ上で実行させると、思い通りには動かない。書かれた通りにしか動かない。この事を実感することに、大きな教育的価値がある。

設計通りに動くプログラムが出来たら、それが最初の意図通りの機能を持つものであるか、評価する。使ってみると、不満足な点が見つかるものである。ここで、また問題の再定式化が行なわれ、設計変更、実現、評価と進めることになる。

3．プログラミング教育の内容

プログラミング教育は、問題の定式化から設計、実現、評価の全過程を体験すべきものであるが、最初からこの順番で進められるわけではない。プログラムというもの自体と、それを記述するプログラミング言語について習熟する必要がある。

3．1 プログラムの基本要素:連接、繰り返し、条件分岐

まず最初に学ぶべきことは、プログラムはやるべき仕事を時間の流れにそって記述するということである。ここで、コンピュータは書かれた事を実行するだけであり、それ以上でもそれ以下でもないことを実感させなければならない。このことは、表現としての文学とは全く異なる方法論を問題としていること意味している。文学では、記述されたこと自体より、それを読んだ人の心にどんなイメージが生じるかが重要である。コンピュータ・プログラムは、それが実行されるコンピュータ上での働きを問題とするのに対して、文学は読者が認識した事実が引き起こす心の動きを問題とするのである。

このように、記述の目的は違うが、その方法論は基本的には変わらない。正確な記述

をどのように達成するかを論じた「理科系の作文技術」を、文学者の井上ひさしは、文学にも役立つ方法論として評価している[2]。

繰り返し機能の導入にあたっては、時間にそっての仕事の記述が同じ記述の繰り返しとなる例を示してから、それが簡潔に表現できる手段として繰り返しの機能を示すのがよい。この際、同じ繰り返し記述に対して複数の繰り返し構造が可能であることを示しておくとよい。使う言語によっては、可能な繰り返し構造全てを実現できるとは限らないからである。例えば、条件文の中に入力文が書けるＣ言話と違って、Ｐａｓｃａｌの while 文では、入力文を繰り返し文の前と、繰り返しの対象の最後の２箇所に書かなければならない。こうした場合、不可能な構造を実現しようとして初心者は苦しむことが多いが、元の単純な繰り返しに立ち戻って表現してみれば、どのような繰り返し構造が可能であるかに気づくからである。こうした経験は、プログラミング言語の表現論に関する議論としても、よい経験となる。

分岐構造の導入は、その意味も分り易く、使い方も直接的で比較的容易である。しかし、条件が複雑になってくると、それを分岐構造の入れ子で表現するのか、条件式を複合条件で表現するこという２つの方法を検討する必要が生じる。どちらの方法をとるかは、様々な条件を考慮して、総合的に決めなければならない。

入れ子構造を使えるのは、分岐構造に限らない。時間にそった記述である連接構造、繰り返し構造も使うことができる。この三つの構造は入り口が一つ、出口も一つという共通の構造を持っているからである。入れ子構造が、構造化プログラミングにおける複雑さに対抗する手段であることを強調する必要がある。

３．２ 手続き（関数）とパラメータ

プログラムを書き出すと、小さな機能を実現するにも、相当の知的労力を費やさざるを得ないことが体験できる。一方、そうした労力も慣れるに従って、あまり苦にならなくなってくると、作れるプログラムの大きさも、段々大きくなってくる。

しかし、一つのプログラムは、２０行程度以上長く書いてはいけない。これが可能となるためには、手続き（関数）の概念を導入しなければならない。手続きは、ひとかたまりの仕事に対して名前をつけ、その名前を呼ぶことによって、手続き全体が実行できる機能である。この機能によって、プログラムの部品を作ることができることになる。部品の階層構造を形成することによって、巨大なプログラムも、短かい記述の積み重ねによってその構造を定義できる。

部品としての手続きには、作業の材料となるデータを渡し、その結果のデータを受けとる機能が必要となる。これには手続きの引数と呼ばれる機能が用いられる。引数には材料となるデータを与えるだけの値引数と、材料となるだけでなく、結果も受け取れる変数引数がある。

大きなプログラムを作るとき、それをどのように手続きに分割していったらよいかは、易しい問題ではない。どのように分割しても、動くプログラムは作れるが、分割の仕方によって、その後のプログラムの価値が大きく変わる。

役に立つプログラムを作ると、もっと役に立つものにしたいと、機能の増強を図りたくなる。こうした作業を行なうときに、手続きの分割方法が悪いと、作業が大きな影響

を受けることになる。こうした経験を経て、手続きの分割方法に関する判断力が養われるのである。

4．協同作業の方法論としての手続きとコメント

プログラミング教育の方法として、グループ作業でプログラムを作らせるのは教育的価値が大きい。グループのメンバー間で確実な意志疎通を図らなければプログラムは作れない。メンバー間のコミュニケーション能力が、作っているプログラムの出来として、直ぐに評価されることになる。

4．1 協同作業の手段としての手続き

プログラムを協同作業で作ろうとすると、手続きの概念が重要となる。設計をしっかりしておけば、手続きを用いることにより、各自が独立してプログラミングを進めることができる。変数の有効範囲、引数といった概念が、実用的な意味を持つことになる。こうした概念を理解させる最も良い方法が、協同作業であると言える。

しかし、こうした協同作業が可能となるためには、問題の定式化や、設計が出来る能力がないと難しい。少くとも、グループのメンバーの中に、こうした能力がある人間がいなければならず、他のメンバーも、彼の設計結果を理解する能力がなければならない。

4．2 作業目的を記述するコメント

協同作業を行なう時に、決定的に重要なのはプログラムにコメントをつけることである。コメントには手続き全体につける見出しコメント、プログラムの塊につけるブロック・コメント、1行全体につける行コメントがある。これらのコメントを適切につけることは、日本語表現の課題として優れているだけでなく、プログラミング作業自体にも大きな影響を与える。例えば、ブロックをどのように構成するかは、高度な判断を伴う作業となる。また、3種類のコメントをどのように使い分けるかも、易しい問題ではない。

コメントとして書くべきことは、その対象とするプログラム部分がどんな仕事をしようとしているかという目的である。このことを指導しておかないと、書かれたコードを日本語に直すだけで済ませてしまう。このようなコメントは全く役に立たない。

一方、目的を書くということは、決して易しい作業ではない。やろうとしている仕事の本質が問われるからである。この意味でも、コメントを書かせることは、表現教育として意味がある。

4．3 目的に基づく設計

見出しコメントとブロック・コメントの間には、前者として書かれた目的を、実現手段として分解したものが後者になるという関係がある。このような分解作業は、実はプログラムを書いてから行なうのではなく、設計として前もって行なうべきである。このような設計法として、ＨＣＰチャートを用いた設計法[3]が有効である。この設計法に従って設計を行なえば、見出しコメントとブロック・コメントは設計が終った段階で出

来上っており、それに従ってコードを書いてゆけはよい。

5．プログラミング言語として何を用いるか

プログラミングの入門教育に、どのような言語を用いるかは慎重な検討を要する。従来はプログラミングは職業課程の学生に教育されていたため、実用言語が用いられてきた。しかし、実用言語は職業人が使い易いように設計されているため、初心者には使いにくく、不必要な作業を強いられることになる。

5．1 教育用言語と実用言語

例えば、C言語を用いると、多様な表現が一つの機能に対して可能であり、どれを使ってプログラムを書かせるかは、慎重な検討が必要となる。教育用に設計されたPascalを用いれば、こうした問題はずっと小さくなる。

実用言語を教育用に用いた場合の問題点として、未熟な学習者が書いた間違ったプログラムも、解釈が可能だと、（書いた人の意図とは違って）書かれた通りに実行されてしまうことがある。こうした虫の発見という作業は表現教育としての価値はほとんど無いが、学習者は長時間を費やさざるを得なくなる。

高校段階で行なう教育は、基礎的な能力を養成し易いPascalやLogoを用いるべきである。こうした言語を用いて基本概念を修得しさえすれば、実用言語の修得は容易である。

5．2 教育用言語 MIND

教育用言語として注目すべき言語として、日本語でプログラムが書けるMINDがある[4]。これは、制御用プログラミング言語として米国で開発されたFORTHを日本語化したものであるが、語順が日本語と同じく、「何を」「どうする」という順で書ける所に特長がある。西欧語の語順で書かなければならないLOGOに較べて、日本の子供には修得し易いという研究結果が報告されている。

実は、日本語の語順は、時系列処理に最も効率的であることが知られている。プログラムに書かれた数式は、逆ポーランド記法と呼ばれる、日本語と同じ語順の記法に変換されてから、処理される。この形式の表現では、カッコを使う必要はなくなり、順次記述にそって処理を進めて行くことが出来る。

逆ポーラント記法は時系列処理には適しているが、読みにくいという欠点がある。この形式の言語であるFORTHやAPLが普及しなかったのは、この点に問題があったからと考えらる。しかし、日本語であれば、少くとも日本人には読みにくさは無くなる。

GUIの普及とともに、西欧のソフトの指定方法も日本語式になった。従来は、西欧語の影響を受けて、「どうする」「何を」の順番だったものが、言語の束縛を逃れたGUIでは、「何を」「どうする」の順番で指定するように変った。これは、日本語式の指定順序が人類一般の通用する合理性を持っているからだと考えられる。合理性は、母国語の語順より強いのである。こうした理由から、MINDは最良のプログラミング言語である可能性がある。

6．さいごに

高等学校における教科として「情報」を考えるとき、その中核となるべきプログラミング教育について、その意義と内容についてコンピュータ科学の立場から議論した。実際に動くプログラムを作成する能力を育成することと同時に、どんな目的でそのプログラムを作ろうとしているのかという議論と、それを実現するために行なう設計について、日本語表現能力か重要であることを指摘した。プログラミング言語の文法を教えることは、プログラミング教育の目的ではなく、その手段にすぎない。問題の定式化、設計、実現、評価という一連の過程を体験させることが、プログラミング教育の中心であり、これを協同作業として行なわせることにより、教育効果を高めることができる。問題定義や設計に日本語が重要な役割を果たすだけでなく、プログラミング言語自体も日本語が使われる可能性を指摘した。

参考文献

[1] Tom van Weert(ed.):Informatics for Secondary education, UNESCO, 1994.
[2] 井上ひさし：文芸時評、朝日新聞（夕）、1981年10月26日
[3] 長野宏宣（監修）：ＨＣＰチャート、電気通信協会、1993年.
[4] 片桐明：日本語プログラミング言語Ｍｉｎｄ基本文法、リギーコーポレーション（横浜）、1992年.