N-029

初心者教育用プログラミング環境に関する一考察

Consideration to a programming environment for beginners

桑原 悟†
Satoru KUWAHARA

１．背景と狙い

１．１　背景

情報処理学会の日本の情報教育・情報処理教育に関する提言2005[1]では，国民全体に対して何らかのプログラミング教育が必要であるとしているが，一方で，一般には，プログラミングは，専門的であり，高度に知的であり，素養のある人材が行うものであると理解されている。

実際に，大学でのプログラミング教育の場では，多くの研究者，教育者から指摘があるように，挫折感を味わい，プログラミングの学習に消極的になる学生も少なくない。

プログラミング教育に関する研究としては，初中等教育でのプログラミング教育を目的とした言語や環境について，既存の実用言語への移行というよりは，プログラミング概念の習得の成果に関していくつかの報告がある。[2)3)]

また，大学でのプログラミング教育に関しては，広く用いられている既存実用言語の大学における教育法に関する取組みも報告されている。[4)5)6)7)]

また，大学での初心者向けプログラミング教育用の言語に関しては，「若葉」[8)]があり，初心者には不要な機能の排除，１種類だけの制御処理記述，単一の数値データ型，初心者には意味を説明し難い必須記述部分の排除の観点から，新しい言語仕様と処理系の設計と実装を行っている。

Nigari[9)]は，学習者の将来のJavaへの移行を意識し，その上で，初心者には「おまじない」と説明するような部分，変数の利用に先立ち書かなければならない宣言文が不要であるなど，初心者向きのプログラミング言語として設計されている。Nigariのプログラミング環境では，テキストエディタのウィンドウでキーボード入力を行ってソースプログラムを作成する。

プログラミングの難しさを軽減するものとしては，PEN[10)]がある。PENでは，プログラミング言語として，特別な説明なしに入試にも用いられているという理由から，DNCL[11)]及びTUATLE[12)]を採用し日本語でのプログラミングを志向している。PENのプログラミング環境では，ソースプログラム入力フィールドの下にプログラム入力支援ボタンを設け，条件分岐，繰返しなどの定型部分を自動生成し，キーボード操作を減らすことができる仕組みをもっている。

このように，さまざまな研究がなされているが，たとえば，初等教育でグラフィック関連のプログラムで興味を刺激し，慣れさせることでプログラミングの概念教育を行っても，実用言語への将来の移行を同じプログラミング環境で行うことはできない。さらにターゲットとなる実用言語が複数の場合は，それぞれ別のプログラミング環境に移らなくてはならず，そのそれぞれは優れた面をもっていても，プログラミング環境の変化という負担を学習者に強いることになる。

また，初心者向けに提案されているプログラミング言語も最初から完全無欠のものを創出することは難しく，変更や改良は当然なされるが，このためのシステムの変更の負荷は相応のものとなると考えられる。

１．２　狙い

今後も教育の場面を絞って新たな教育用プログラミング言語とプログラミング環境が研究されることは，この方面の発展にとって望ましいが，本研究では，興味を刺激するグラフィック関連の命令からなる言語も，実用言語への将来の移行を意識した初心者教育用言語も同じプログラミング環境で実現することを狙いとする。

そのために，プログラミング環境の設計は，次の二つの基本方針で行う。

- プログラム入力支援の機能を充実させる。この機能は，特定の言語専用ではなく，定式化された言語仕様を定義する情報を与えることで，その言語に合った構文要素などを提示する機能とする。
- 独自の翻訳及び実行系をもたず，各教育用プログラミング言語で書かれたプログラムを，既存プログラミング言語に変換[a]する機能をもつ。翻訳及び実行系は，既存プログラミング言語のものを利用する。

ここで，入力支援機能については，プログラミング人口の拡大を妨げている“難しさ”を抽出し，機械的手段で排除できる可能性のある要素については，その手段を検討する。

排除できない要素，すなわち，現時点では，人間の知的生産活動によらなければならないものについては，プログラミング教育の立場から，このプログラミング環境で使える教材を作成し教育，訓練に寄与する方向で取り組む。

２．難しさの抽出

２．１　プログラミング作業の概観

人間が行う，知的生産活動としてのプログラミングは，おおよそ，次のような手順を踏むと考えられる。

まず，使用するプログラミング言語の文法知識と過去のプログラミングの経験から，構想を立て，次に，これに基づいて，コーディングを行う。

コンパイラを用いる言語の場合，次に，コーディングしたソースコードをコンパイルし，実行する。インタプリタを用いる言語の場合は，翻訳と実行が合わせて行われる。

† 新潟国際情報大学　情報文化学部

a)コンパイルを意味する「翻訳」と区別するため「変換」という

この段階で，大抵の場合，特に入門者の場合は，実行完了にいたらず，幾つかのエラー情報がコンパイラ又は実行系から与えられる。人間のプログラミング活動としては，このエラー情報を見て，構想又はコーディングの段階へ戻り，ソースコードを修正する。この道筋は，大抵の場合，複数回繰り返される。

修正後，コンパイルと実行の段階で，エラーが無くなった後，そのプログラムが目的とした機能を備えているかどうか，テストを行う。この段階で，目的の機能が実現されていない場合，やはり，修正が必要になるが，これは，コーディングの段階での修正の場合と，構想段階への戻りが必要な場合，及び，その中間的な修正の場合があり得る。これらの各修正は，幾つかに分類できる。

２．２　文法違反の発見

プログラミングにおいては，コンパイラ，インタプリタ又は実行系が，エラーメッセージとして表示するものから，そのエラーの原因を特定し，修正する作業が必要である。このとき，エラーメッセージが示すものが，直接修正すべき個所を示す場合と，エラーメッセージからは，修正すべき個所が直接読み取れない場合とがある。

現在のプログラミング言語のコンパイラ，インタプリタ又は実行系の構造から考えて，人間にとってのエラーメッセージの的確さの度合いは，後者の状況を多く起こさざるを得ないといえる。

後者の場合，通常，プログラムをする人間は，そのプログラミング言語に関する他の知識などから修正すべき個所を類推すること，試行錯誤的なコーディングの変更を行って，結果を観察すること又は，これらの組合せ及びその繰り返しを行って，修正すべき個所を特定していく。

学習者は，初期の学習課題においては，すでに教材に示されたコーディングをそのまま入力して動かす課題，それを参考にして，指示された限定的な変更を加えて動かす課題も与えられる。

被験者がこの段階の作業を完成できない，すなわち修正すべき個所の特定にいたらない場合も見うけられる。また，指導者の助言や指導者による修正個所の特定の過程を見せても，別の類似のエラー個所で，やはり，この段階の作業を完成できない場合も観察される。

２．３　意図と記述の齟齬の発見

前述のエラーとして表示されるものは除き，プログラムが，プログラミングした人間の意図とは違う振る舞いをする種類の誤りがある。これには，言語によってさまざまな場合があり得るが，名称，記号の誤記，抜け又は過剰及び，記述位置の誤りなどがあり得る。

たとえば，括弧の位置，終止符の抜け，符号の誤った記述などが，編集中の誤操作などにより引き起こされる例，あるいは，変数を予め定義する必要の無いプログラミング言語で，変数名の綴りの一部を誤ると，別の新しい変数として扱われ，意図した結果を得られない例などがこれにあたる。

エラーとして表示されないので，プログラム作成者は，プログラムの振る舞いから誤り個所を特定することになるが，作成者は，正しく記述した積もりになっていることもあり，すぐには発見されない場合が多い。

一文字の綴り誤りを発見するのに，プログラムの実行結果とコーディングリスト，デバッグのためのツールを用いて誤り個所を特定することになり，これもプログラミングの難しさを構成する一要素になっている。

２．４　構想の誤りの発見

プログラミングの学習において与えられる課題は，初心者に対する極めて初期のものでない限り，例題をそのままコーディングするのではなく，学習者は，これまでで得たプログラミング以外の知識，たとえば数学の公式などと，プログラミングの学習で得た知識すなわち，利用できる演算の種類，用意されている関数，手続き型の言語においては，場合分けや繰り返し処理の記述などの知識を用いて，課題解決の構想を立てる。

当然，この段階においても，完成できない場合や誤りを生じる場合がある。これは，明らかに前述の難しさとは独立であると考えられる。

２．５　論理の誤りの発見

前述の“構想の誤り”は，ある一つの課題を考えたとき，プログラムする人間と使用する言語との組合せによって，結果が異なることが考えられる。被験者Aが，言語Xでプログラミングを行う場合には誤りを犯さず，同じ被験者Aが，言語Yで同じ課題の解決のためのプログラミングをする場合には誤りを犯す，あるいは，完成にいたらないという事態が起こり得るからである。これは構想に使う言語の知識の誤りに原因をもとめることができる。

これとは別に，言語によらず，被験者によってだけ誤りが起こる状況を考える。これは，被験者の論理的思考の能力に関連するものと考えられ，知的生産活動を分類する立場からは，このことは，論理的思考を別の一要素と考えてよいことになる。

しかし，構想の誤りとこの論理の誤りを，実際のプログラミング作業とその結果から，完全に識別することは難しいと言える。

３．各要素の検討

ここでは，２．で上げた四つの要素について，機械的に排除できる可能性のある難しさであるかどうかを検討する。

３．１　文法違反に対する検討

前述のように，コンパイラや実行系の表示するエラーメッセージは，多くの場合，プログラミングをする者が直接知りたい情報を与えてはいない。コンパイラを作成する立場から見ると，コンパイル時又は実行時のエラーメッセージを，プログラマが直接知りたい内容にすることは困難である。

そこで見方を変え，コーディングが文字入力による自由記述の形態で行われることに，この種の誤りが発生する原因があると捉えた場合，これに対しては対策が比較的取りやすい。

自由記述を制限し，選択肢からの選択をすることでプログラミングを行う環境がその一つの答えとなる。この環境を使用することにより，予約語と同じ名称の変数の使用，関数の引数型や演算においての型の不整合なども防止することができる。

既存のプログラミング言語に対し，このような環境を用意する場合は，詳細な検討が必要となるが，教育に用いる既存プログラミング言語のサブセットを考えると，この要素は，プログラミングから排除できる可能性のある難しさである。

３．２　意図と記述の齟齬に対する検討

意図と記述の齟齬に関しては，完全に排除することはで

きない。しかし，３．１で述べたプログラミング環境は，この要素に対しても有効である場合がある。特に，変数名の綴り誤りは，選択肢からの選択を行うことで，入力誤りを回避できる場合がある。

３．３ 構想の誤りに対する検討

プログラミングにおいての構想の作業は，プログラミングの本質の一つであると考えられ，これを排除することは難しい。

しかし，特に初心者にとっては，言語のマニュアルなどの書籍を常備することの煩わしさや，それを参照する手間といった，この要素に付随する事象に関しては，プログラミング環境における適時の表示やオンラインマニュアルなどで軽減することができる。

３．４ 論理の誤りに対する検討

論理の構築は，プログラミングの本質又は根幹をなすものと理解でき，この要素を排除することはできない。したがって，この要素は，プログラミング教育の教材で訓練すべき要素と考えることができる。

４．プログラミング環境

ここでは，前章で検討した，難しさの排除のための具体的機能と１．２で示した，言語変換機能について述べる。

４．１プログラミング入力支援機能

前章の結論では，ソースプログラム作成において，自由入力を極力制限することが一つの目標であった。

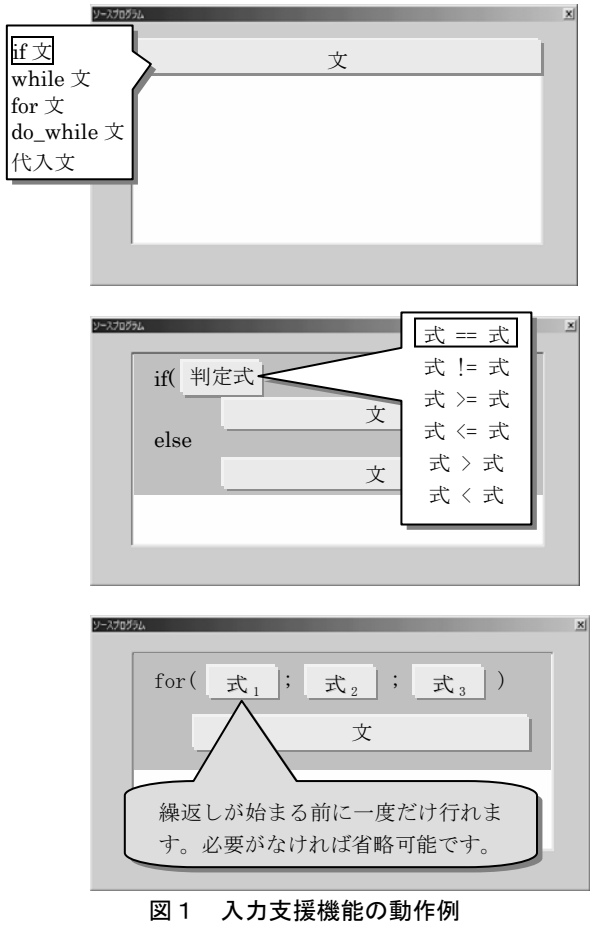

図１　入力支援機能の動作例

ここでは，Ｃ言語の機能サブセットで，かつ初心者に不要な記述を省いたプログラム言語を想定してその機能を説明する。図１の上の画面は，最初の入力の場面である。文を入力するフィールドを右クリックすることで，そこに入力できる文の種類の選択肢が現れる。ここで，if文を左クリックすると，図１中の画面のようにif文の構成が自動的に入力される。

この構成中には，判定式と二つの文を記述すべきフィールドが表示されており，判定式のフィールドを右クリックすると判定式としてこの言語で記述できる形式の選択肢が表示される。この中から，選択肢を選び，さらにこの種の作業を続けていき，プログラムを作成していく。

図１下の画面は，for文を例に，フィールド上にマウスをもってきたとき，表示されるそのフィールドの説明を示している。

これらの選択肢と説明の表示は，次の言語定義情報[b]から生成される。

BNF

文:= if文 | while文 | for文 | do_while文 | 代入文
if文:=“if” “(” 式 “)” 文 〔 “else” 文 〕
for文:=“for” “(” 式1 “;” 式2 “;” 式3 “)” 文

構文要素説明[c]

〈for文/式1〉“繰返しが始まる前に一度だけ行われます。必要がなければ省略可能です。”

〈for文/式2〉“この式を評価した値が0（ゼロ）でない間，文が繰返されます。省略はできません。”

〈for文/式3〉“繰返しが終わる時に一度だけ行われます。必要がなければ省略可能です。”

４．２言語変換機能

１．２で述べたように，このプログラミング環境は，独自の翻訳，実行系をもたない。

図２は，Ｃ言語を将来の習得ターゲットに据え，ヘッダのincludeやmain関数関連の記述など初心者には説明し難い部分を排除した教育用プログラミング言語を想定した変換例である。

ヘッダのincludeやmain関数関連の記述が追加される。構文は，簡単なルールで変換されることが分かる。

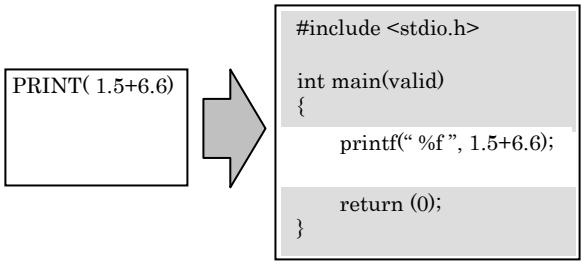

図２　変換の例

b）これらは，C言語のものではなく，想定した教育用プログラミング言語のものである。

c）構文要素説明の◇内は，索引であり，どのBNFのどの位置の要素の説明であるかを示す。

ターゲットとなる言語があるということは，構文などを大きく変えることは望ましくないので，自ずと類似の構成となる。このことは，ターゲットとなる言語がある場合，その言語への変換のルールは，複雑にはならないことを示しており，ここでの設計方針は有効であると言える。

また，グラフィック機能の命令を中心としたものなど，初心者の興味を刺激する種類の教育用プログラム言語の場合では，Javaでグラフィック機能を利用することは難しくなく，命令の引数の順序や数などを教育用言語のものに合わせるなどのメソッドを用意すれば，もともとこの種の教育用言語の仕様は大きくないので，実現は容易である。

これらの変換には変換ルールと追加記述の情報が用いられる。

４．３その他の機能

図３は，このプログラミング環境の構成を示す。ソースプログラムの保存と読出しの機能，キー入力機能，予約語の誤用や文法違反となるような型の違う変数間の代入などを検査し，警告を表示するチェック機能も実装する。

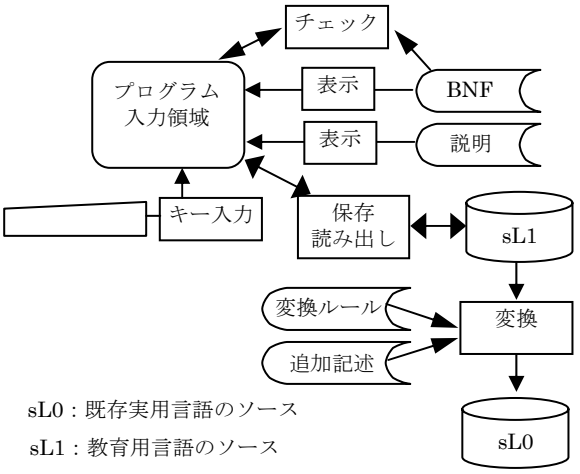

図３　プログラミング環境の機能構成

５．今後の課題

今後は，実装のための設計の細部の確認，調整を終了し，作成を急ぐ。また，提案した入力支援機能は，プログラミングの初心者のみならず，キーボード入力をさらに少なくすれば，バリアフリーのプログラミング環境への発展が期待できるので，社会的にも意義の大きい方向への適用も視野に入れる計画である。

参考文献

1) 情報処理学会情報処理教育委員会:日本の情報教育・情報処理教育に関する提言2005,情報処理学会(2005).

2) 兼宗進,中谷多哉子,御手洗理英,福井眞吾,久野靖:初中等教育におけるオブジェクト指向プログラミングの実践と評価,情報処理学会論文誌(トランザクション),Vol.44,No.SIG13,pp.58-71(2003).

3) 情報処理学会情報処理教育委員会:教育用プログラミング言語に関するワークショップ2006報告集.

4) 武内亮／佐藤匡正,プログラミング教育における合理的評価法,情報処理学会自然言語処理研究会,Vol.2004,No.93,pp.153-159(2004).

5) 塚本邦昭:プログラミング教育環境支援システム開発に関する研究,情報処理学会情報システムと社会環境研究会,Vol.1994,No.61,pp.39-46(1994).

6) 河村一樹,斐品正照,徳岡健一:文科系向けプログラミング教育支援システムJPADetの設計と開発,情報処理学会コンピュータと教育研究会,Vol.2001,No.122,pp.9-16(2001).

7) 佐野洋:大学の文科系学部におけるプログラミング教育の試み,情報処理学会コンピュータと教育研究会,Vol.1998 No.102,pp.41-48(1998).

8) 吉良智樹,並木美太郎,岩崎英哉:初心者入門用言語「若葉」の言語仕様と処理系の実装,情報処理学会論文誌(トランザクション),Vol.40,No.SIG10,pp-28-38(1999).

9) 長慎也,甲斐宗徳,川合晶,日野孝昭,前島真一,筧捷彦:プログラミング環境Nigariー初学者がJavaを習うまでの案内役,情報処理学会論文誌(トランザクション) ,Vol.45,No.SIG09,pp.25-46(2004).

10) 中村亮太,西田知博,松浦敏雄:プログラミング入門教育用学習環境PEN,情報処理学会コンピュータと教育研究会,Vol.2005,No.104,pp.65-71(2005).

11)大学入試センター：センター試験用手順記述標準言語ーDNCLー,平成15年度センター試験 試験問題評価委員会報告書,pp.258-259(2003).

12)中森眞理夫, 中條拓伯,小谷善行,辰巳丈夫,金子敬一,並木美太郎,品野勇治:平成18年度入試に向けての「情報」試行試験の実施報告(2),情報処理学会第46回プログラミングシンポジウム報告集,pp.173-180(2005)

13) 桑原悟:プログラミングにおける知的生産活動要素,新潟国際情報大学紀要,No.10(2007).

14) 岡田英彦,旭敏之:PC初心者ユーザのためのGUIナビゲータ/カバーの開発と評価,情報処理学会論文誌(ジャーナル),Vol.43,No.6,pp.2006-2016(2002).