はしがき

プログラミングは、これまで専門学部や学科の教育内容であった。しかし、情報化への対応として高等学校以上の学校においても、数学あるいは一般教科の中でプログラミング教育が行なわれ、平成５年度からは中学校技術・家庭科に「情報基礎」という新領域が設けられ教育内容の一部にプログラミングが盛り込まれている。

これまでプログラミングの評価は、プログラミングが専門領域の問題解決の手段として位置付けられ、一定水準を超えればよしといった観点で結果や成果に基づいて行なわれてきた傾向がある。しかし、前述のような状況のもとでは教育目標の達成や学習内容の理解を明確に評価していくことが要求される。

プログラミングの様相を観察してみると、学習者がさまざまな方法によって問題解決に取り組んでいる。すなわち、プログラミングにおいては多くの知的スキルと技能を必要とする。このような学習においては、作成過程を把握し指導や評価に反映させていくことが重要と考えるが、作成過程に視点を置いた評価の事例は見当らない。

そこで、本研究においてはプログラム作成中のキー入力データを全て収集し、そのデータを再び言語プロセッサに自動的に再入力することによりプログラム作成過程を自動的に再現できるシステムを用いて、その作成過程すなわちエディティング過程、プログラミング過程、デバッギング過程を明らかにしつつ、プログラミングにおける評価項目を抽出して、作成過程に基づく評価法を開発することを目的とする。

情報教育に携わっている教師は同じプログラムであっても作成過程に差があることを経験的に知っている。しかし、現在作成過程を把握するツールがないことから結果だけで判断したり、評価問題により知識を問うことによって評価を行なっている。本研究においては、消去された部分まで再現し作成過程を全て捉え、そこでの評価法を開発しようと試みた。評価研究の新しい切り口となれば幸いである。

研究組織

研究代表者：　中野　靖夫　（上越教育大学学校教育研究センター　教授）
研究分担者：　南部　昌敏　（上越教育大学学校教育研究センター　助教授）
研究分担者：　田中　　敏　（上越教育大学学校教育学部　助教授）

研究経費

平成5年度	900千円
平成6年度	600千円
計	1,500千円

研究発表

1．近藤智嗣、中野靖夫：Logoプログラム作成過程の分析,電子情報通信学会技術研究報告,ET94-65,1994-09,pp.31-36
2．前田恵三、中野靖夫：Ｃ言語のプログラミング過程の分析,電子情報通信学会技術研究報告,ET94-65,1994-09,pp.37-44
3．田中 敏：コンピュータ・プログラムの作成に関する教育的評価次元の構成，上越教育大学研究紀要，第14巻第2号，1995-3
4．小岩寿之、松島健一郎、横田亮宏、南部昌敏：プログラミング中に発生するエピソードとそのカテゴリー化，日本教育工学会研究報告集，JET 95-2 1995-3

口頭発表

1．近藤智嗣、中野靖夫：Logo学習におけるキー操作の分析，日本教育工学会第9回大会講演論文集,1993-10,pp.310-311
2．近藤智嗣、中野靖夫：Logoプログラミングのカテゴリ分析,教育工学関連学協会連合第4回全国大会講演論文集(第2分冊),1994-10,pp.523-524

目　次

1. 研究の背景

コンピュータを使用するには、ソフトウェアすなわちプログラムが不可欠である。このため、プログラムを記述する言語の開発が進められ、機械語、アセンブリ言語、高級言語、ビジュアル言語と発展してきた。一方、コンピュータの普及に伴い、多くのアプリケーションソフトウェアが必要になり、これらの言語を使用してプログラミングが行なわれている。

歴史を遡ると、大学、企業等で汎用コンピュータを使用できるようになった時代にはプログラミング言語、主としてFORTRAN、COBOL、ALGOL等の教育に力が注がれた。当時はアプリケーションソフトウェアは普及しておらず、コンピュータを使用するにはプログラム開発をさけて通ることはできなかったのである。

その後、学校教育において、プログラミングは専門学部や学科、商業、工業高校などで情報処理教育の中で取り上げられてきた。しかし、プログラム作成はあくまで手段であり、目的は情報処理、シミュレーション、数値計算など高次の内容あるいは問題解決を目指していた。つまり、プログラミングが「できる」、「できない」ことより、目的の解決が「できた」「できなかったか」が問題であったのである。

プログラミングは演習課題を解くような方式が主流を占め、学生はトレーニングを積み重ねながらプログラミング技法を習得していった。従って、個人の発想を生み出す事より既存の知識を習得する事に力が注がれたといえよう。

1971年、アメリカのintel社が4ビットのマイクロプロセッサを開発して以来、半導体技術の技術革新により8、16、32bitの製品が次々に開発され、これらを使用したコンピュータは、小型化、コストダウン、高性能化の道を辿っている。現在、コンピュータは個人で使用するような状況になり、名称もパーソナルコンピュータと呼ばれている。パーソナルコンピュータが登場したときには、それを動かす手段として、初心者向けの言語であるBASICが附属していた。コンピュータを使用するには、やはりプログラムが必要であった。しかし、コンピュータが高性能になり、より使いやすくするために、各種のアプリケーションプログラムが開発され、コンピュータが多目的に使用できる状況になってきた現段階で、ようやくプログラム言語はシステムから切り離されたのである。

コンピュータの普及と情報化社会の到来により、学校教育もその対応に迫られ、カリキュラムの改訂やコンピュータの設置が進められた。大学の一般教育や高等学校の数学などの教科の中でプログラムは教材の一つとして取り上げられるようになった。また、平成元年度の学習指導要領改訂においては、中学校技術・家庭科に「情報基礎」が加えられ、内容の一つにプログラミングが盛り込まれた。通産省の報告等では、ソフトウェア技術者が不足するという内容が述べられ、産業界は情報処理技術者の育成を望んでいたが、これらの教育は情報処理技術者やプログラマの育成を目指しているわけではない。

プログラミングは与えられた問題あるいは解決しようとする課題の要求を定義し、プログラムのモジュールや手順を決め、図式化する。そして、プログラムをコーディングし、テストを行ない評価・修正するという、長いスパンの知的作業である。

プログラミングによって、まず、コンピュータを動かす手段としてプログラムの必要性、

入力を受付、処理し、結果を出力させることで、コンピュータの役割、機能、処理モデルを理解させる事ができる。また、論理的な考え方を育成したり、構造的な処理方法を体験させることができる。さらに、言語プロセッサとの対話により個別の学習が成立するなど、一斉授業とは異なる教育が実現する。特筆することは、学習としてのプログラミングは、新しい方法の創造あるいは、知的な生産活動であり、情報化時代の能力育成に適合した活動といえる。

このようなとらえ方をすれば、情報化時代を迎えた現時点でプログラムは学習の対象になったといえる。かって、プログラミングは問題解決の手段であったが、プログラムは学習の目的になったのである。従って、プログラミングの教育内容、教育方法、教育評価の検討には、教育学、認知科学、心理学、工学、情報学等からのアプローチが必要になってきたのではなかろうか。

2. 研究目的

学校教育の教科の中で行なわれるプログラミングは、大型のプログラムの設計、開発と異なり問題解決過程を直接コーディングしていくことが多い。プログラミングは、与えられた問題を分析しそれを解決する方法が分かることが前提であるが、プログラミングの様相を観察してみると、学習者がさまざまな方法によって問題解決に取り組んでいる。作成手順がきちんとまとめられ、整然とコーディングする者もいるが、逐次試行しながらコーディングする者もいる。その作成過程は、問題の解決手順が構成出来たか、プログラミングに関する理解があるか、言語プロセッサに関する知識はあるか、課題に取り組む興味や関心は十分であるかなどによって左右される。すなわち、プログラミングは多くの知識と技能を必要とする学習であり、作成者の論理展開、試行、模索等の知的ふるまいが作成過程に表出してくる。

これまで、プログラミングに関する認知の解明や教育効果を高めるなどの目的で、プログラミングに関する分析、評価研究が展開されている。例えば、宮地はFORTRANの算術代入文について誤答分析を行ない誤答原因を9つに分類している。渡辺らはPascalの学習終了後に、プログラムを読み取らせ mental runningで数値計算を行なわせ、その時間から理解の程度を測定している。江木らはCOBOLプログラミング中に異常終了したリストを収集しその誤り傾向を調査している。これらの研究は、学習の終了後、あるいは異常停止が発生した時点で分析、評価しており、静的データを扱っている。

一方、作成過程に着目し動的データを分析した研究も進められている。その一例として岡本らが診断助言型のＩＴＳを用いて分析した事例があるが、この研究では、メンタルモデル解明のため机上でコーディングを行なわせながら作業内容について口頭で説明させ、これをテープレコーダで録音しオンラインプロトコルを採取している。従って、実験という制約された環境下におかれたデータ収集で、かつ言語プロセッサと対話した時のありのままの作成過程ではなく技能面や記述、編集、消去、修正等の状態遷移あるいは記述内容を変更したときの操作データは得られていない。しかし、前述のようにプログラミングを

知的ふるまいとして捉えればその作成過程のデータすなわち動的データを全て追跡し分析・評価することは認知科学や情報教育の視点からも意義ある事と考える。

そこで、本研究においては、プログラム作成中のキー入力データを全て収集し、そのデータからプログラムの作成過程を掘り起こし、作成過程に視点をあてたプログラミングの評価法を開発することを目的とする。

3. 研究の方法

1)プログラム作成過程のデータ収集

この研究では、プログラム作成過程のデータ収集を行なわなければならない。被験者に測定を意識させずにデータ収集の自動化をはかるには、コンピュータプログラムの活用が最適である。コンピュータプログラムが作成過程のキー入力を全て収集すれば、これが作成過程の原データとなる。そのデータを逐次解読していけば作成過程の全てを把握できる。

しかし、キーデータを追跡していくだけでは、カーソル移動や特殊キーが叩かれたときの言語プロセッサの対応の状況を分析者が理解することは困難である。そこで、収集したデータを自動的に言語プロセッサに再入力し、動作を再現することができれば、分析者がデータ収集の場にいたことと同じ状況となる。これを実現するコンピュータプログラムを既に開発しており、本研究では、そのプログラム（コンピュータ操作過程の再現システム）を使用した。以下に動作の概要を示す。

システムは幾つかのモジュールにより構成されている。図1の矩形内の上段に役割、下段にファイル名を示す。通常のシステムにおいては、キー入力があったとき、破線で示すように、本体の割込みルーチンがキーデータをとりこみ、キー入力リングバッファに格納する。そのデータはＢＩＯＳ、MS-DOSを介してアプリケーションソフトウェアへ転送される。このシステムにおいては、実線で示すように割込みルーチンに代わって、キーボードドライバ(KBHIS.SYS)が、キーデータをキー入力リングバッファに格納していくとともにヒストリ記録用テーブルにキーデータと操作時間を転送する。あるキーの操作時間とは、直前のキー操作から、あるキーが操作された時点までの時間である。使用しているアプリケーションソフトウェアを終了する際に、ヒストリ記録用テーブルのデータを保存プログラム(SAVEHIS.COM)によって外部記憶装置に記憶させる。そのデータは出力プログラム(PRHIS.COM)でＣＲＴ、プリンタあるいは外部記憶装置に出力することが可能である。動作を再現させる場合には、再現プログラム(REPLAY.COM)を起動する。すると、外部記憶装置に記憶されていたデータは、再びヒストリ記録用テーブルに呼び出され、キーボードドライバは再現モードに切り替わる。ヒストリ記録用テーブルのデータは、逐次、キーボードドライバ、ＢＩＯＳを介してアプリケーションソフトウェアに転送される。このようにして、あたかもキーボードから入力を受け付けているかのように動作させ操作を再現する。

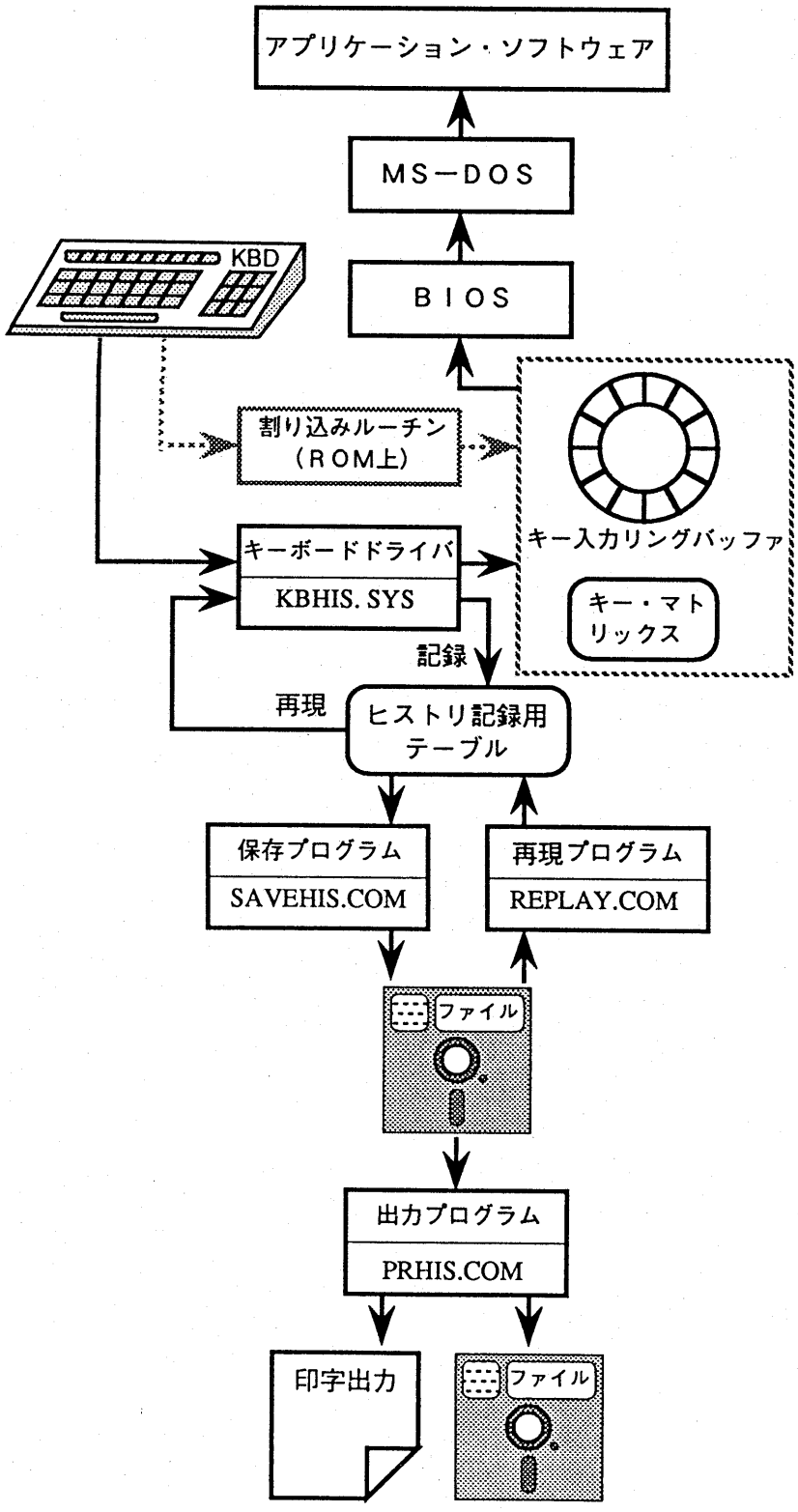

図１　システムの動作モデル

2)評価事項、項目の抽出

プログラミングの中で学習者、教師がそれぞれの立場で、「やりたいこと」「やろうとしていること」に対して、実際にはどのように「やってきたか」「やっているか」という作成過程を適切な観点から具体的に明らかにする必要がある。その活動の過程や結果が意図したものに対して、どのような意味や価値をもっているかについての解釈や判断を与え、次の行動や目標を決定するのによりよい指針と方略を示すことが評価の課題になる。

プログラミングは問題を分析し、解決の仕方を見いだし、その解決方法を設計し、手順を記述し、それをテストし評価・修正していく長いスパンの知的作業である。従って、学習の進度により具体的学習内容、学習活動項目は変化していくので、評価の観点や評価項目もこれに対応させる必要がある。

ところで、本研究は完成したプログラムで評価するのではなく、プログラム作成過程に視点をおいた評価法を開発することである。プログラムの作成過程を再現し観察・分析すると、いかなる評価事項や項目あるいは基準が抽出されるのであろうか。プログラミングは言語プロセッサと対話しながら手続きを記述していく学習（知的作業）であり、論理的に評価事項や項目をあげる事は可能である。しかし、適切な評価を行なったりフィードバック情報を生成するには、プログラムをどのように「作成してきたか」「作成しているか」を具体的に明らかにする必要がある。例えば、「言語を理解していない」、「キー入力に誤りが多い」などの評価には、「‥‥という状況で」「‥‥‥とした」という事実やデータを示すことである。

そこで、まずプログラムがどのように記述されたのか、その過程、すなわちプログラムの構築方法を明らかにする。また、作成中に発生した出来事（これを「エピソード」と呼ぶ）を取りだしていく。「エピソード」が多種多様であればカテゴリ化などの技法を使用する。このような方法は、評価者がプログラム作成時に同席して逐次データを収集しなければならず、データ記録や多人数のデータ収集が困難であるため、これまで行なわれていなかった。本研究ではコンピュータ操作過程の再現システムを使用することによって、プログラム作成中に発生した事実を取りだすことが可能となった。このようにして取りだされた事実やデータを基にして、評価の観点や項目を構造化していく。

4．作成過程の1事例＝個人のプログラム作成過程

資料(pp.21-47参照)にPascalでプログラムを作成したキー操作履歴データとその解釈を示す。

作成者は教育学部2年生で、Pascalの講義･演習(90分)を7回学習しており、プログラミングの基礎知識は習得している。また、90分の自学自習を10回行なっている。事前学習の中でｅのグラフのかきかた、三角関数表の作成に関するサンプルプログラムが配布され、前者のプログラムを入力し、実行している。

作成者は、TAN(X)のグラフを書かせることを試みている。課題は教師から与えられたのではなく、自らが新規の課題に取組んだものである。

言語プロセッサは、TURBO Pascal ver.2.0を使用している。このシステムのエディタは、Word Starと同じコマンド体系で、現在使用されているエディタとキー操作は異なるが、データ収集時点では習熟度が上がっておりキー操作に負担は無いと考える。

資料の左側ページはキー操作履歴データ列即ち作成過程であり、右側ページはその解釈である。作成過程はデータを読みやすくするため、ワープロで編集作業を加えている。

プログラムは逐次記述され、最初のコンパイル時までの変更、修正は少ない。このような記述方法は、①事前に問題が分析されプログラムの設計が済んでいる②サンプルのプログラムを加工していく③再度プログラムを作成する場合に実現できる。ここでの作成者は、②③の両者に該当している。

入力時のキー操作のミスおよび文法上のミスが少なく直ちにコンパイルに成功し実行できた。ここまでに発生したエピソードを、表1に示す。

表１　最初のコンパイルまでに発生したエピソード

操作の区切り	エピソード
(004)	1文字削除
(004),(005),(009)	カーソルキーの使用
(008)	TABキーの使用
(009)	;(セミコロン)の追加
(013)	型名の変更
(014)	,(コンマ)の削除
(024)	数値データの変更、charデータの入力ミスと修正
(027)	数値データの変更
(031)	入力ミスと修正
(033)	命令文の修正
(034)	変数のスペルミスとその修正
(038)	隣接キーをたたき、その修正

しかし、目的の結果が得られず、作成者はプランナーからマニュピレータになり、操作の区切り(048)から(340)までプログラムのデバッギングを繰り返す。(048)から(184)までのデバッギング過程の分析結果を表2に示す。

エピソードは数値データの修正が多い。数値データの修正は、Y軸(縦軸)の長さ、Y軸の刻み線の間隔、X軸上の補助軸の間隔、グラフの振幅、三角関数の引き数を変化させる。作成者は、演算結果の状態を確認しながら上記のような修正を行なっている。(076),(080)においては、tanのグラフを正値のみに変更している。これは、数学の知識とは整合性が無いように思われるが、何らかの手立てで問題を解決しようとしている。長いデバッギング過程が続き、(148)(152),(156),(160)は連続して三角関数の引き数を修正している。ここでの評価は知識や論理的なことより、心的な分析、評価が必要になると考えられる。

これ以後も、デバッギングが続行されたが最終的にプログラムは完成していない。この事例では文法上のミスが無いため、言語プロセッサからのメッセージはない。結果を分析し論理的な考察を行ない修正しなければならないのであるが、その場しのぎと思われる試行錯誤が連続している。つまり、論理的な誤りに属するエピソードが多発したといえる。

三角関数の角度はラジアンに変更しなければならない。しかし、この手続きの入力、追加は行なわれなかった。つまり、三角関数に関する知識習得がなされていなかった、あるいは参考として配布された資料が活用されなかったのである。この調査を行ったとき、教師は知識を教え込むのではなく、自らが知識獲得を行なえるようにするため資料の配布に留め、講義で問題について説明していない。授業内容が十分でないと、このような事例が多く発生すると考えられる。作成者は授業で学習した内容はプログラミングを行なっており、教えられたものは身につけるという態度で望んでいたが、三角関数は未知の問題解決であり既習内容では解決出来なかったのである。

表２　デバッギング過程に発生したエピソード

操作の区切り	エピソード
(048),(052),(064),(084) (104),(120),(124),(128)	・数値データの修正 Y軸の長さを修正
(064),(116),(120),(136) (144)	Y軸の刻み線の間隔を修正
(048),(176)	グラフの周期を修正
(056),(060),(140),(144) (164),(168),(172)	X軸上の補助軸の間隔を修正
(056),(072),(088),(100) (108),(112),(160),(180) (184)	グラフの振幅を修正
(072),(084),(096),(148) (152),(156),(160)	三角関数の引き数を修正
(068)	・命令の変更 関数をsinからtanに変更
(076),(080)	・方法の変更 グラフを正値のみに変更

このプログラム作成以前にcos(X)のグラフを描かせているが、近似計算によって描かれた曲線で妥協し完成したと認識している。また、プログラム修正中に、三角関数を数値化して検証するようなことは行なわれていない。つまり、プログラムの開発方法に誤りが有ることに気付かず、自己の方法に妥当性を見出したと考えられる。従って、本課題を作成することは不可能であった。

上述したように作成過程を分析すればプログラムの作風、すなわち構築過程やキー入力、エディタの活用状況、プログラム作成過程およびデバッギング過程で発生したエピソードや特定の状況で作成者が何を試みたかが明らかになる。これまでの研究では作成過程の解明についてはバグの研究を除けば殆ど着手されていない。また、ここで明らかにしたデバッギング過程は文法上の誤りが無くバグの研究でも扱いにくいのである。

データ収集は観察によって可能であるが、コンピュータ操作のような多量のデータ処理は技術的に困難な点が多い。本研究ではコンピュータプログラムによる非反応測定法によってデータ収集を行ない作成過程の再現し、提出されたリストでは見ることのできないエピソードの取りだしを実現したのである。

5. 発生するエピソード

プログラム作成中に発生するエピソードは学習の進度、キー操作技能、学習者の理解度、課題、教授内容・方法、システムの機能等に起因する。そこで、エピソードを抽出しこれを分析していけば新たな評価法が確立されると考える。

では、どのようなエピソードが発生するのであろうか。また、その発生箇所や発生数はどの程度の数になるのであろうか。以下に幾つかの言語プロセッサによるプログラミング中のエピソードの発生について考察する。

1)Pascal

資料では、ある個人の作成過程の全てを示した。もう一例は発生したエピソードを順次記述していく。

この事例は教育学部2年生が90分、3回の講義、演習を受けた後、2次関数の解を求める(ただし、係数a,b,cはキーボードから入力する）課題を作成する過程である。

・判定条件の記述の誤り
・判別式Dの符号による場合分けを追加
・解を求める式を変更
・X:= -c/b →X:=(-c/b)　式に（　）を追加
・変数の宣言に1文字追加
・wrute →write　予約語の修正
・b^2 →b*b　　算術式の修正
・squr →sqrt　　関数名の修正
・end文の追加
・writeln文の追加
・文の削除　8行
・write文に桁数の指示
・read文を3行削除
・write文の桁数の指示を修正

・else文の挿入

以下省略

上述のようなエピソードの記述のみでは、観察者や分析者以外は、およその傾向が把握できるものの詳細な状況把握はできない。これは、文字情報と画面（２次元的パタン情報）による情報の形態の相違によるもので、二種類の情報処理を行なうのは困難点が多い。「どのような状況で」という状況説明は文字情報で記述すると大変手間のかかる作業になる。したがって、構築過程を把握し、数的分析あるいは時系列分析を行なうには１文ごとに作成状況を分析し、ある時点での解釈、分析、評価を行なうには状況に応じてプログラムリストを提示していく必要がある。資料のように全過程を記述すれば、観察者以外でも分析評価が可能である。

また、プログラム作成過程は、「ある状態」で「どのようにした」とか、「以前どのようなエピソードが発生したか」によって作成過程が変化する。すなわち作成過程には、流れ、文脈、ストーリーのような連続性が存在するため、作成の再現を直視しシステムとの対話を読解していくプロトコル分析のような手法が必要になる。これは認知工学からの接近となろう。

2)Logo

口頭発表-1は、A、B二つの中学校の3年生(それぞれ30名、37名)を対象にしたLogoプログラミングにおいて発生したエピソードを分析した。データ収集以前の学習時間は、A中学校０時間、B中学校６時間である。

A中学校はLogoの学習以前にBASICを学習しているが、①［ ］の入力時に｛ ｝（ ）＜ ＞等を入力した者が68%、②[DEL]を使用しないで[→][BS]を使用する者が73%③不要な[INS]キーをたたく者が60%いるなど、キー入力の誤りや消去技能の未熟さ、不要キーの使用などに関するものが発生する。また、forward　100の入力においては、命令のスペルミスが33%、forwardの入力のみでリターンキーをという書式に合わない記述が47%発生している。このように特定の短い操作を分析対象にした場合は、エピソードの分類、計数、分析は容易である。また、repeat文で三角形をかく課題では、線分の方向を変える角度の指定が一回目では全員が誤っていた。ここでは、角度を120としなければならないが、60と入力した者が47%であった。repeat文を使用しない者が20%いるが、これらは数値データの欠如、問題の解決方法の誤り等プログラミングの知識、技能に関するエピソードである。

B中学校は、6時間の学習を行なっており、正方形とそれを回転させる課題を作成させたところ、[：]の入力時に[；]を入力した者が10%いるが、これ以外にキー入力に関するエピソードはあまり発生していない。回転させる角度の設定時に正方形の命令を変更した者が20%存在した。この学級は、キー操作には習熟しており、プログラムの記述法の一部に論理的な誤りがあるが、授業や評価に反映させるような状況は発生していない。

口頭発表-2は、中学校2年生17ペアが10時間目の学習時に4問の課題を作成する際に発生したエピソードのカテゴリ分析である。また、この実験の詳細は研究発表-1にまとめられている。

課題1は2つのプログラムを参考にしてタートルを１方向に動かす手続きの作成である。

発生したエピソードは、入力行ごとに分析したところ、入力、修正、加工に分類できる。入力、修正、加工における出現比は175:23:5である。作成者は正しい命令を逐次記述していく手法でプログラミングを行なっており、修正、変更のエピソードは少ない。入力を正誤で比較すると161:14で、誤った文の入力は極めて少ない。また、正しい文の入力のうち改行以前に修正された文は67である。従って、この集団のプログラミングの作風は、命令の入力を正しく修正しながら記述していく方法をとっているといえよう。

3) C

プログラミングの学習が進み、学習者の状況を観察していると作成方法が幾つかのタイプに分類できる。研究発表-3ではプログラミングに習熟した8名の工業高等専門学校4年生を以下に示す4つのタイプに分類し、その作成過程を分析した。

A) じっくり問題を分析してからコンピュータに向かう。プログラムの作成は早い。

B) 少しずつプログラムを入力しながら、途中でよく考える。

C) プログラムを段階的に考え作成し試行錯誤で完成させる。行き詰まった時には、じっくり考えずにプログラムを変更しては試すことを繰り返す。

D) C言語の文法とアルゴリズムが十分身についていないため、作成途中で教官の助けを求める。

これらのタイプは教師が、机間巡視や授業経験から決定した。プログラムの作成過程の再現により、4タイプの作成者の作成過程を具体的に記述することが出来た。資料で示したように個人の作風やプログラム構築方法を明らかにしている。また、30秒以上の思考時間、HELPキーの使用時点、カーソル移動数や削除数からは第3者が作成状況を推測することが可能である。

発生したエピソードはキー操作に関するものは少なく、間違いの発生とその対処法に関するものである。

再現システムでは、キーストロークと操作時間が直読できる。これらは作成過程の評価項目の一つになると考えられる。しかし、思考時間と操作時間はVTRやアイカメラ等で作成過程をモニタリングするなどの必要性があり、それを欠如した本研究では精度は低い。

コンパイルと実行の頻度の高い作成者のエピソードを見ると、コンパイルエラーはif文の条件が逆などの簡単な論理的ミスによることが多い。コンパイルエラーの文章が適切でないときには、原因を見つけるために試行錯誤を繰り返している。エピソードの発生はプログラム作成環境について影響を受けている面もあり環境についても考慮する必要性がある。

①カーソル移動キーの入力が多いのは、エディタで一度に見ることのできる行数が18行と少ないためでマルチウインドウのエディタで変数宣言部と実行部の記述を同時に表示して入力を行なうことができれば、カーソル移動が減ると考えられる。

②文の終りにセミコロンを入力しないと、その先の文で理解できないエラーメッセージが表示され、原因の分からないままの試行錯誤が各所で発生する。コンパイラが指摘する行とエラーメッセージの表示が改善されることが望まれる。

③C言語はデータの抽象化が弱いため配列同士の代入や比較が演算子を用いて行なえず直

感的に理解しにくく学習の妨げになっている。また、文字列の型がないのに変数宣言では一見文字列定数の代入ができるようになっているため、混乱を招いている。

以上幾つかの言語の作成過程やエピソードについて述べてきた。しかし、言語と学習進度、学習者の関係の組合せを考えると、この研究期間に全てのエピソードを抽出したり、計量化することは困難である。従って、ここでは幾つかの事例の報告にとどめたが、およそ以下に示すことが言えよう。

a)初期段階では、キー操作やエディタの使用法、命令の記述法に関するエピソードが発生する。たとえば、キータッチミス、シフトキーの操作、エディタの使用法の適切さ、スペリング、プログラムの書式の誤り等である。

b)まとめの段階では、プログラム作成以前の問題解決でのつまずきやアルゴリズムに関するエピソードが発生する。たとえば、問題の解法、再帰手法の活用、入出力の条件設定等である。また、この段階ではプログラミングに関するエピソードよりデバッギングに関するエピソードが多くなる。

c)課題を設定すれば特定のエピソードが事前に想定できる。しかし、作成者は予測出来ない操作を行なうことがある。

d)エピソードを計量化し、その数値に解釈、判断を加えれば、プログラミング技術の傾向、特徴をつかむことができる。

e)長いエピソードは幾つかのエピソードに分割しないと解釈が困難になる。そこで、たとえば誤文と気づかずに入力し、その後ある文を入力中に誤文に気づき、誤文の行に移動し挿入、修正を行ない復行したような場合には、①誤文の入力②他の文への移行③文の入力とする。つまり、分析を文単位および操作単位で行なうという手法がエピソードの抽出を容易にする。

f)エピソードの発生はプログラム作成者の個人的な要因と学習内容、学習方法、言語プロセッサ等の環境によって異なる。

6. エピソードのカテゴリ化

5で示したようにエピソードは言語、学習者、学習進度、課題等により異なる。発生した多種多様のエピソードは類似のものや系列化できる特徴を有しているものもある。このような事例の集合体はカテゴリ法により分類、整理、分析できる。

プログラム作成は、①問題の解決方法を発見し②その解決手順を言語で表現し③その手順を評価し修正・改善していく知的作業である。従って、作成過程の最上位カテゴリには次に示す３つの作業過程を設定する。

1)エディティング
2)プログラミング
3)デバッギング

最も低レベルのエピソードはエディティングにおいて発生する。すなわち、入力に関する知識・技能（キー操作やエディタ操作)に関連するエピソード群である。プログラミング

(プログラムの記述)に属するのは、問題解決に必要な知識・方法、言語プロセッサの知識に関連するエピソード群である。デバッギング(プログラムの評価･改善)は、一旦作成したプログラムを動作させ，目的の機能や性能を有しているか点検し、修正・改善していく作業である。ここに属するのは、誤りの発見、目的との不一致の対応策、言語プロセッサの知識に関連するエピソード群である。通常、2)、3)を統合したものが、プログラミングと言われている。しかし、プログラム作成過程において、研究発表-3に述べられたようにプランナとマニピュレータでは処理方針や作業方針が異なり二面性を有しているところから、本研究では分割した。

上記の作業過程において、それぞれのカテゴリは、知識、技能、方法、時間という中項目に分類できる。更にエピソードを分析しサブ･カテゴリに分類した一例を表3に示す。

表3　エピソードのカテゴリ

作業過程	エピソードの内容		
エディティング	知識	E	エディタの操作に関するもの コマンド、カーソル移動 消去、複写
	技能	EM	ミスタッチなどキー操作に関するもの 入力モード、シフトキー 隣接キーのタッチ
	時間	ET	キーを見つけるまでに要した時間 予約語、変数の入力において
プログラミング	知識	P1	予約語等の綴りに関するもの スペリング
		P2	文法に関するもの 書式 構文 改行、記号
		P3	言語プロセッサに関するもの コマンド、状態の理解
	方法	PM1	問題を解決するための方法に関するもの
		PM2	プログラムの構築方法に関するもの､アルゴリズム
		PM3	訂正（挿入、変更、修正、削除）
		PM4	他行への移行
	時間	PT	思考時間
デバッギング	知識	D	予約語等の綴りに関するもの スペリング
		D1	文法に関するもの 書式 構文 改行、記号
		D2	言語プロセッサに関するもの コマンド、状態の理解
	方法	DM1	問題を解決するための方法に関するもの
		DM2	プログラムの構築方法に関するもの､アルゴリズム
		DM3	訂正（挿入、変更、修正、削除）
		DM4	他行への移行
	時間	DT	誤りに気づくまでに要した時間

エディティングにおいては、コマンドの使用法、カーソルの移動、消去や複写の方法に関するエピソードが発生する。キー操作技能とは入力モード設定、シフトキーの使用法隣接キーのタッチなどキー操作に関するものである。これらはプログラムを入力する上での知識、技能である。

プログラミングの知識に関連するエピソードは、第一に予約語や命令語のスペリングに関するものであり、隣接キーのタッチと区別した。第二は書式、構文、記号に関するエピソードである。書式は、命令の記述形式に誤り、例えば、語間にスペースキーが脱落しておりこれを訂正したようなエピソードである。構文は、IF文、繰返しのモジュールがPascalで定義されている構文図のような形式上の体裁に整えているかが問われる。記号は、BASICにおける改行やセミコロン、コロンの使用法、C言語における記号群の使用に関するエピソードである。ところで「；」と「：」は隣接キーで且つ役割が大きく異なるので、いずれのカテゴリに計上するかが判断の分かれるところである。第三は言語プロセッサに関するエピソードである。これはエディット、コンパイル、実行、HELP時のコマンド使用やキー選択に関するものや言語プロセッサの特定の状態における対応である。

プログラミングの方法に関連するエピソードのうち問題を解決するための方法である変更とは、例えば、利息計算を一覧表で出力する方法を元金、利率、年数を読み込んで、計算し出力するように変更したような場合である。プログラムの構築方法に関するものやアルゴリズムは、多量の逐次計算を繰返しにするなど、構文上のエピソードである。訂正（挿入、変更、修正）は数値データや命令の訂正（挿入、変更、修正）に関するものである。他行への移行はある文を作成中に他行に移動しなんらかの操作を行なった場合で他のエピソードに連結する。

デバッギングは、一度実行した結果に誤りがあり、これを修正していく過程である。従って発生するエピソードはプログラミングと同様のカテゴリに分類できる。

カテゴリは分析視点を設定すれば更に細項目を設定できる。また、言語プロセッサおよび評価課題を設定した場合は、分析用の細分化したカテゴリを作成すれば詳細な操作事実を把握することができる。これらのカテゴリは評価項目の一つとなり、その度数を計量して評価することが可能になる。

研究発表4においては、BASICプログラミングのエピソードを抽出しそのカテゴリ化を試みた。短期大学1年生が90分の授業を6回受講した後、基礎的な知識で解決できる3問の課題を作成する際の作成過程のデータを収集しエピソードを分析した。プログラミング過程はプログラムの入力を１行単位に区切り(改行が行なわれるまで)、文の正誤を確認しながらエピソードを14個のカテゴリを分け、このカテゴリを用いて集団のプログラミングの分析、評価を行なった。また、デバッギング過程において実行、入力、リスト表示を項目に加え、ある作成者の時系列カテゴリから、その特徴を明らかにした。さらに課題別にカテゴリ分析を行ない修正内容を分析し度数を計量した。

なお、BASICはインタプリタ型の処理系でコマンドの使用に関しては上述したようにエディット、プログラミングに厳密に区分けすることは困難である。

個人の作成過程は、４で示したように個人ごとに履歴を追跡すれば詳細に分析、評価することができる。一方、エピソードすなわち事実を数値データに変換すれば、一般に行なわれているように個人や集団の状態を数量的に分析、評価することが可能になる。さらに、エピソードをカテゴリ化すれば、カテゴリ別に計量することができる。また、エピソードに、カテゴリコード、個人コード、学年、言語、学習時間、キーワード、プログラムリスト等の属性を付していけば、エピソード・データベースが構築できる。

カテゴリ分析を行なえば、個人、集団により発生したエピソードの頻度、状態遷移、時系列等が明らかになる。また、個人、集団の二つのデータの比較が可能になる。これらのデータを評価的視点で分析を進めることにより、学習者、教授法、教育内容、システムの評価が可能になると考える。

7. 作成過程に視点を置いた評価

プログラム作成の作業に、従来の学習・思考・問題解決に対する種々の評価視点がそのまま適用できるならば特に問題はない。もちろん、これまでの多くの既存の評価視点も用いるべきであるが、それと並行してプログラム作成それ自体に固有の評価視点を用いる必要があるかどうかが重要な問題となる。

現在のところ、プログラム作成に固有のシステマティックな評価の枠組みは、本プロジェクトが取り組み始めているようにようやく開発の途についたばかりであり、今後とも丹念なエピソードの収集と信頼性のある分析を持続していかなければならないであろう。

この章では、そうした開発研究を方向づけるうえでの基本コンセプトについて、ここまでの知見に基づいた試案を提出してみたい。

(1)言語使用者と情報処理者

主要なコンセプトは、「言語使用者」と「情報処理者」との対比としてまとめられると思われる。この対比は問題解決スタイルの違いであり、その違いは課題遂行場面に存在するメディアとデバイスの違いに由来する。たとえば、言語使用者はいわゆるペーパー・アンド・ペンシル・タスクを遂行する者として想定することができ、これに対して、情報処理者はディスク・アンド・コンピュータ・タスクを遂行する者と考えることができる。

順番に特徴づけをおこなってみる。

まず、言語使用者は通常のペーパー・アンド・ペンシル・タスクの遂行において、知識の形成と運用をすべて言語を介しておこなわなければならない。したがって、当然のことながら言語使用者は、自分の中に言語を所有すること、および自分の中から言語を生産することが要求される。すなわち、言語使用者は自分自身がたくさんの語彙をもち、優れた概念体系をつくり、あざやかなことばの操作を示さなければならない。一言でいえば言語使用者はその人が「有能である」ことを要求されている。

これに対して、情報処理者はディスク・アンド・コンピュータ・タスクの遂行において「有能である」必要はない。たくさんの語彙や知識は、その人自身が保有しなくてもディスク（ＦＤ，ＨＤ，ＣＤ，ＬＤ，ＭＯＤなど）に保存されている。いわゆるデータ・ベー

スあるいは知識ベースは、情報処理者にとって外在的なものである。情報処理者は必要に応じて、それらにアクセスし、適宜、読み出せばよい。また、そうして読み出された情報はコンピュータが実行するプログラムの中で運用される。したがって、何らかのアイディアの生産はプログラムに依存しており、ここでは、情報処理者は生産に関わっているというよりも、むしろ「生産に関わるプログラム」の選択に関わっていることになる。いわゆるプログラム・ライブラリィまたは作業メニューは、情報処理者の中に保有されている必要がなく、これもまた情報処理者にとっては外在的なものである。

要するに、卑近な例示をすれば、言語使用者が問題とすることは自分が「頭がいい」かどうかということであるが、情報処理者が話題とすることは、「いい」ハードウェアが出たとか、「いい」ソフトウェアがあるとか、ということである。

言語使用者と情報処理者の違いは、このように課題遂行スタイルの違いにとどまらず、興味・関心のあり方の違い、ひいては心的発達の違いにもつながると思われる。

(2)情報処理者としてのプログラム作成者

以上の対比に関して次のように考えることもできるであろう。

たとえば、言語使用者と情報処理者の違いについていえば、アプリケーション・ユーザーは確かに情報処理者であるといえるが、彼らにソフトを提供するところのプログラム作成者は情報処理者というよりも言語使用者に近いのではないか、と。

おそらく、この問題への解答が、プログラム作成過程をどのように評価すべきかについての基本的枠組みを構築するであろう。

筆者の試論的解答は次のとおりである。プログラム作成者は、やはり自分自身が有能化してゆく言語使用者ではなく、知識と能力の外在化に向かう情報処理者である。この考え方の根拠は、単純に、プログラム作成もまたディスク・アンド・コンピュータ・タスクにほかならないというところにある。

その特徴は、第一に、プログラムを作成するための語彙と文法は宣言的ではなく手続き的なものであるという点にある。手続き的知識は作成者本人の認知体系を再構造化するものではない。手続き的知識は手段を増やすだけであり概念を深めるものではない。いわゆる “how to” に関するものであり、“what is” に関するものではない。たとえばプログラム作成の作業において、物事の処理に関するフローチャート式の作業イメージが強く膨らむことはあっても、それは処理の仕方についての認識であり、処理される内容についての認識ではない。

第二の特徴として、作成されたプログラム自体も、作成者を有能化するものでないということがあげられる。すなわち、作成されたプログラムは作成者の頭の中に有意味性をもって保有されるのではなく、メモリ・メディア上にたんなる「ファイル名」をもって記録されるものである。もしも、このプログラムが自然言語と同等の語彙と構文をもっているのであれば、そのプログラムはそのままの形態で作成者の内側に表象され、保存されるであろう（たとえばプログラム中の “ＲＩＮＧＯ” という変数がリンゴを思い浮かべるときに頭の中にあらわれる）。しかし現実にはすべてのプログラム言語は、いわゆる解釈言語であり、機械語と自然言語の翻訳コードとしてのみ存在する。すなわち知識を表象する言

語ではない。この意味において、作成されたプログラムはその作成者本人にとって外在的なものにとどまっている。これは言語使用者ではなく、やはり情報処理者が遭遇する事態である。

プログラム作成過程の特徴は、このように「手続き的過程」および「外在的生産物」ということにまとめられるであろう。

(3)プログラム作成の作業傾向の特質

以上のプログラム作成過程の特徴づけから、プログラム作成の学習者に関する固有の評価視点を予想することができる。

再度、言語使用者と情報処理者の対比を援用すれば、言語使用者は宣言的（概念的）知識と内在的生産物（自分の知性）に取り組み、情報処理者またはプログラムの作成者は手続き的知識と外在的生産物に取り組む。前者の、概念と知性の規準は「正しいこと」であるが、後者の、手続きとプログラムの規準は「うまくゆくこと」である。

この対比をいくつかの同義の対比にいいかえると、「プランナー」対「マニピュレータ」（ミンスキー in ブランド，1988）、「インテリジェンス」対「インフォメーション」、「体系的処理」対「場当たり処理」（後２者とも田中，1994）となる。

筆者がおこなった後掲の「研究発表－３」では、この対比に対応するプログラム作成作業の諸傾向が見いだされている。詳しい結果については当該ページにゆずるが、特にプログラム作成過程の評価視点を検討するうえで特に重要な知見は次のとおりである。すなわち、コンピュータ・プログラムに関する理解度が高い学習者ほど、プログラムの表記・内容に関する形式的明確化（たとえば前後行の変数のケタ揃えなど）をどうでもよいものと考えるようになり、また、課題の要求とは異なる作業（プログラムの使い勝手をよくするなど）に自発的に取り組むようになるということである。

この傾向は、プログラム作成の学習者が自らの内在的なインテリジェンス（知性）のあり方を志向しているのではなく、外在的なインフォメーションのあり方（プログラムの機能や利用しやすさ）を志向していることを示唆していると考えられる。

こうしたプログラム作成に固有の作業傾向を、当の評価視点の枠組みの主要部分とすることを考慮しなければならないであろう。全体の枠組みとしては、後掲の「研究発表－３」で抽出された他の２つの作業傾向因子を、上述した主要傾向の前段・後段に配置した以下のようなものが考えられる。

(4)プログラム作成過程の評価視点

以下の評価視点の書き方は、学習者の困難をチェックすることを目的として、ネガティヴな標記内容としてある。したがって、プログラム作成への習熟度をチェックする目的をもつ場合には、標記内容をポジティヴに読み替えていただきたい。

①新規事項への着手の回避

これはプログラム作成の導入段階によく見られる傾向である。典型的には、キーボードの操作にあらわれる。たとえば、プログラムのコマンドを書きまちがえて訂正するときに、

デリート・キーやインサート・キーを使うのをおっくうがり、まちがえた部分だけでなくコマンド全体を書き直す。さらにもっと初歩的段階では、手のホームポジションや各キーの位置を覚える意識的努力をせずに、いつまでも目でさがして１本の指で押すという動作にとどまって、「キーボードを見ずにキーを押せるようになる」とか「連続したキータッチをこころがける」とかの新規の学習事項へ移行しないことなどが予想される。

②プログラムの機能よりも表記の重視

この段階は、キー操作よりも進んで、プログラムそのものに対して大きな注意が振り向けられるようになったが、それにもかかわらず、いまだプログラムの機能よりも、その書き表し方などの表面的事項にとらわれてしまう段階である。

通常の自然言語であれば、表記を首尾一貫した形式にすることはそのまま知識の整理につながるが、コンピュータ・プログラムの表記にはそれは望めない。変数を忘れないためとか、実行が移る行を確かめるためとか、プログラミングの誤りを少なくしたりバグの発見を容易にしたりするための、たんに備忘録的効用しか認められない。したがって、プログラムがあまりに繁雑にならない限り、プログラムの表記形式よりも、実際にそのプログラムがどんな仕事をするのかという機能的側面に集中することがこの種の作業における心理資源の配分法として適切であるといえる。

③その場しのぎ・不拡大方針

この段階は一応の課題の要求にこたえた段階である。かりに言語使用者の作業スタイルであれば、この段階で何らかの新しい知識が形成されているはずである（学習による生産物は何らかの形で内在化しているはずである）。したがって、ここで終了しても（多かれ少なかれ）獲得物がある。

しかしながら、プログラムの完成は作成者本人の知識を更新するわけではない。課題の要求を満たすプログラムができあがった時点で、たんに「つくり放し」の状態に陥るおそれがプログラム作成作業にはつきものである。すなわち、作ったプログラムは一定の心理的所産として内在化されるような性質のものではなく、ディスクに保存して終了するだけである。そこで、教育的な評価視点としては、さらに使い勝手のよさとか汎用性を追求するような目標をこの種の学習課題に固有の目標として掲げておくべきであろう。実際、コンピュータ・プログラムの理解度の高い者はそのような問題意識を自発的に生じる傾向のあることが観察された。

④探究・発展の回避

これは新規の課題を求めるかどうかに関しての評価視点である。標記内容は治療的な観点に立つものであるが、もちろんポジティヴに読み替えて「探究・発展の志向」とするほうが形成的評価につながる。

この段階は、あたえられた課題が終わった後に自己学習へ移行する段階である。ポイントは、自分自身でもっと課題を持続しようとするかどうかということにある。この点は、言語ベースの課題においても、ディスク・アンド・コンピュータ・タスクにおいても、重

要な評価視点であることは多言を要さないであろう。

すでに述べたように、以上の評価視点のうち(2)と(3)が、プログラム作成過程に固有の評価視点として想定されたものである。その前後の(1)と(4)は、特にプログラム作成過程に固有というものではない。

注意すべきは次のことである。すなわち、プログラム作成という学習課題は、それ自体として取り上げるべきではないし、したがってそれ自体として評価すべきではない。そうすることは、この章の基本コンセプトである言語使用者と情報処理者の違いを強化してしまうであろう。たんにコンピュータ利用という目標からプログラム作成に取り組ませる場合、その作業過程と作業出力の外在化をまねくだけであることは十分に注意する必要がある。「言語使用者であることを前提とした情報処理者」でなければ、プログラム作成は学習者の知性の形成につながらない課題遂行となるのみである。この点、学習課題としてだけではなく研究課題としても、まずもってプログラミング作業の内容を何よりも優先して吟味、選別すべきであることを筆者自身、痛感している。学習者からの採集したデータがどのような評価視点を示唆し、どのような教育目標を示唆するかは、プログラム作成の作業内容をいかに知性の形成に結びつけるか（たんなるコンピュータ・リテラシーの獲得にとどまらず）にかかっている。

(5)今後の課題：情意領域の評価

以上は、プログラム作成過程の認知領域の評価を念頭に検討したものである。これと並んで、当然、情意領域の評価も考慮に入れておかなければならない。実際、言語使用者と情報処理者との対比的コンセプトは、情意面においても異なる特徴があらわれることを予想する。

これまで言語使用者の動機づけの主な要因として、「報酬」「自己」「課題」「他者」の４つが見いだされている。このうち、たとえば、言語ベースの課題遂行は「自己」の有能さに関係することがよく取り上げられてきたが、前述したように情報処理者の課題遂行の過程と結果が当の本人にとって外在的であることによって「自己」要因の媒介が低くなるのではないかと考えられる。たとえばプログラム作成がじょうずにできるようになっても、自己が有能になったというよりは、他人よりもじょうずであるという競争的側面が強調されなければ、動機づけの作用をもたないかもしれない。また、言語使用者の言語はもともとコミュニケーション言語であるため「他者」との協同を自然に仮定することができるが、プログラム作成は基本的にマシンとのパーソナル・インターフェイスを介しているため「他者」要因の強さが相対的に落ちることが考えられる。

こうした情意領域の評価も、言語ベースの課題遂行には見られない、プログラム作成過程に固有の視点を用意しなければならないことは容易に予想されるところである。

文　献

ブランド，Ｓ．　室謙二・麻生九美（訳）　1988　メディア・ラボ　福武書店
田中　敏　1994　心のプログラム　啓文社

8. おわりに

大学の一般教養、高等学校普通科、中学校等で実施されるプログラミング教育はプログラマの育成を目指しているわけではない。言語プロセッサというシステムに対し、プログラム言語という制限された条件で考えを記述していく知的作業を通して、コンピュータを理解させたり知的な制作をすすめる学習として捉えていかなければならないであろう。

本研究は被験者に測定を意識させること無く、コンピュータを使用した非反応法(unobtrusive measure)によって多量のデータを収集出来る方法を適用した。また、このコンピュータプログラムは分析者の望む速度で何度でも作成過程を再現し観察できる。これは本研究の特色と言えよう。この方法は個人の作成過程を詳細に分析できる。まず、プログラムの作風すなわち構築方法がわかる。一方、プログラム作成中に発生したエピソードから入力の技能やエディタ、言語および言語プロセッサ等の理解、活用の方法を評価することが可能である。キー操作時間からはキー操作技能、スペリングを評価できよう。また、エピソードの抽出により、「特定の状況」で、「どのように対応した」かが明らかになる。エピソードは、たとえば命令語を間違えたので直ちに修正したという短いものもあるが、同じ傾向の誤りが何度も発生したり、作成中にたえずカーソル移動があり、いろいろの作業が行なわれることもある。このような長いストーリー性のあるエピソードからは新たな評価視点が見出せると考えるが、長い動的な事実やデータを追跡して評価していくには認知科学や心理学的な接近を加えた学際的な体制が必要になる。

ある集団で発生したエピソードの内容を分析し、その発生度数を計数すれば、集団を評価することができる。しかし、この評価は集団そのものの評価に加え、教育内容、方法、言語プロセッサ等の評価が含まれる。教育の場では教師側の評価が行なわれることは少ない。しかし、全ての作成者あるいは無作為に抽出された作成者について、発生した事実やエピソードの内容とその度数、発生傾向を分析すれば学習環境の評価ができる。

プログラミングの分析、評価は、これまで工学的な視点として取り扱われてきた。本研究においても作成過程を再現し、観察法によりエピソードの抽出や計量からの評価法を試みている。しかし、今後は新しい学力観で示されているような関心・意欲・態度、思考力・判断力・表現力についても評価視点を見出していかなければならない。

プログラム作成は言語プロセッサと作成者の対話であり、この観察と分析はプロトコル分析の一つと考えられ認知工学からの接近が必要になろう。収集したデータはこれからも継続的に分析し、さらに、評価視点を設定した課題による実験も実施する必要がある。ここで得られた評価指針や知見を実際の授業に適用し、その効果や学習者のプログラミングの変容を見ていくことが今後の課題である。

本研究を進展させるにあたり、文京区教育センター教育機器研究部専門指導員 近藤智嗣氏、東京工業高等専門学校 前田恵三氏、山形県教育センター 伊藤憲一氏、秋田短期大学 小山内幸治氏にはデータ収集・分析等で多大なご協力を頂きました。ここに深謝致します。

参考文献

江木鶴子、岡村健史郎、長田一興：COBOLプログラミングにおける初心者の誤り傾向、CAI学会誌 6-3、pp.26-36、1989

江木鶴子、岡村健史郎、長田一興：ゴール／プラン分析法による初心者の作成したCOBOLプログラム誤り分析、CAI学会誌 7-2、pp.80-91、1990

前田恵三、中野靖夫：コンピュータ操作過程の再現システム、日本教育工学雑誌16-4、pp.37-44、1993

宮知功：BASIC入門教育における学習者の変容について、CAI学会誌 8-2、pp.90-98、1991

岡本敏雄、安田恭一郎：C言語プログラミングのメンタルモデルの分析-診断助言型のITSを用いて-、日本教育工学雑誌16-3、pp.119-130、1992

渡辺馨、渡辺祥子、神宮英夫：プログラミング能力の速度としての反応時間、CAI学会誌 9-1、pp33-37,1992

市川忠男、平川正人：かわりゆくプログラミング、共立出版、1994

ジェラルド･M･ワインバーグ／木村泉、角田博保、久野靖、白濱律雄 訳：プログラミングの心理学、技術評論社、1994

T.ギルブ,ジェラルド･M･ワインバーグ／木村泉、米澤明憲 共訳：計算機入力の人間学 共立出版株式会社、1991

海保博之、原田悦子：プロトコル分析入門、新曜社、1994

付　記

この報告書は研究代表者、研究分担者3名による共同執筆により作成した。ただし、「7.作成過程に視点をおいた評価」は、田中敏の個人執筆である。

資　　料

```
Data Count : 4917
Shift key group status ---
   SHIFT: OFF
   CAPS:  OFF
   ｶﾅ:    OFF
   GRPH:  OFF
   CTRL:  OFF
(001) y
(002) e
(003) b : t BS s a n n BS BS BS BS t a n _ x / BS . p a s CR
(004) p r o g r a m SP s a n k a
      k u k a n n BS ↑ ↑ → ↑ ↓ BS ^G s u ; CR
(005) ↑ ↓
(006) SP SP SP
(007) c o n s t SP a m p = 2 0 CR
(008) TAB
(009) r o w m a x = 1 0 ↑ ↓ ^E ^S ^S ^S ; ^X ^D ^D ^D ^S ; CR
(010) BS BS BS BS BS BS
(011) v a r CR
(012) SP SP SP
(013) l i n e : a r r a y [ - a m p . . a m p ] o f SP i n t e g e r
      BS BS BS BS BS BS BS ^G ^G ^G ^G ^G ^G ^G c h a r ; CR
(014) r o w , c o l , i , BS ^G : i n t e g e r ; CR
(015) x : r e a l ; CR
(016) BS BS BS
(017) b e g i n CR
(018) SP SP SP
(019) f o r SP r o w : = 0 SP t o SP r o w m a x SP d o SP b e g i n CR
(020) SP SP SP
(021) f o r SP i : = - a m p SP t o SP a m p SP d o SP b e g i n CR
(022) SP SP SP
(023) l i n e [ i ] : = ' SP ' ; CR
(024) i f SP ( i SP m o d SP 1 BS 5 ^G ) = 0 SP t h e n SP
      l i n e [ i ] : = : BS ' | G ^G ^G ^S ^G ' ; CR
(025) BS BS BS
(026) e n d ; CR
(027) i f SP ( r o w SP m o d SP 1 0 ^S ^S ^G ^G 5 ) = 0 SP t h e n CR
(028) SP SP SP
(029) f o r SP c o l : = - a m p SP t o SP a m p SP d o CR
```

キー入力の解釈

(000) キー操作数、シフトキーの初期状態
(001) include error messages で y を選択。
(002) edit modeにする
(003) ファイル名の入力。tと入力、これを消去。sannと入力し、これを消去。
tan_x/と入力、Bキーで/ を消去し、.pasと入力、改行。エディット・モードとなる。
(004) program sankakukannと入力し、BS、カーソルキーをたたき＾Gで、nを消去、su:と入力、改行。
(005) カーソルキーの操作
(006) インデント(三文字空白)
(007) const amp=20、改行
(008) TABキーをたたく。
(009) rowmax=10と入力、カーソルキーの操作、カーソルを移動し、(007)に;を追加、カーソルを移動し、(009)に;を追加、改行。
(010) BSキーを移動し、インデント
(011) var、改行
(012) インデント
(013) line:array[-amp..amp]of integerと入力、BSキーでカーソル移動、integerを消去し、char;と入力、改行。
(014) row,col,i,と入力、BSキーでもどり、, を消去、:integer;と入力、改行。
(015) x:=real;、改行
(016) BSキーでインデント
(017) begin、改行
(018) インデント
(019) for row:=0 to rowmax do begin、改行
(020) インデント
(021) for i:=-amp to amp do begin、改行
(022) インデント
(023) line[i]:=' ';
(024) if (i mod 1と入力、BSキーでカーソル移動、5を入力し1を消去。
)=0 then line[i]:=:と入力、カーソル移動、' |Gを入力し、:|Gを消去、'; と入力。
(025) インデント
(026) end;
(027) if (row mod 10と入力、カーソル移動、10を消去、5)=0 thenと入力、改行。
(028) インデント
(029) for col:=-amp to amp do、改行

```
(030) SP SP SP
(031) l i n e [ c o l ] : = ' - ) BS ^G ' ; CR
(032) BS BS BS x : = 0 . 5 * r o w ; CR
(033) c o l : = r o u n d ( 1 0 * t BS s i n ( x ) ^G ) ; CR
(034) i f SP ( c o l > = - a m p ) a n d ( c l < = a m p )
        ^S ^S ^S ^S ^S ^S ^S o ^D ^D ^D ^D ^D ^D ^D SP t h e n CR
(035) SP SP SP
(036) l i n e [ c o l ] : = ' * ' ; CR
(037) BS BS BS
(038) f o e BS ^G r SP i : = - a m p SP t o SP a m p SP d o SP
        w r i r ^S ^G t e ( l i n e [ i ] ) ; CR
(039) w r i t e l n CR
(040) BS BS BS
(041) e n d CR
(042) BS BS BS
(043) e n d . CR
(044) ^K ^D
```

(030) インデント

(031) line[col]='-)と入力、カーソル移動、)を消去、';と入力、改行。

(032) インデント、x:=0.5*row;、改行

(033) col:=round(10*tと入力、カーソル移動、sin(x)を入力、tを消去し、);と入力、改行。

(034) if (col>=-amp)and(c1<=amp)と入力、カーソル移動、c1間にoを挿入、カーソル移動、thenと入力、改行 。

(035) インデント

(036) line[col]:='*';改行

(037) インデント

(038) foeと入力、カーソル移動、eを消去、r i:=-amp to amp do wrirと入力、カーソル移動、rを消去、te(line[i]); と入力、改行。

(039) writeln、改行

(040) インデント

(041) end、改行

(042) インデント

(043) end.、改行

(044) エディット終了

プログラム

```
1.        program sankakukansu;
2.           const amp=20;
3.                 rowmax=10;
4.           var
5.              line:array[-amp..amp]of char;
6.              row,col,i:integer;
7.              x:=real;
8.        begin
9.           for row:=0 to rowmax do begin
10.             for i:=-amp to amp do begin
11.                line[i]:=' ';
12.                if (i mod 5) then line[i]:='|';
13.             end;
14.             if (row mod 5)=0 then
15.                for col:=-amp to amp do
16.                   line[col]:='-';
17.                x:=0.5*row;
18.                col:=round(10*sin(x));
19.                if (col>=-amp)and(col<=amp)then
20.                   line[col]:='*';
21.                for i:=-amp to amp do write(line[i]);
22.                writeln
23.             end
24.          end.
```

```
(045) c
(046) r
(047) e
(048) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^S 3 ^G ^X ^D ^D 2 ^G
(049) ^K ^D
(050) r
(051) ^K e
(052) ^X ^D ^F ^D ^F ^F ^F ^S ^S ^S 1 ^G ^X ^D ^X ^X ^X ^X ^X ^X ^X ^X
      ^X ^X ^X ^X ^X ^X
(053) ^K ^D
(054) r
(055) ^K e
(056) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^X
      ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^X ^X ^X ^X ^X
      ^X ^X ^X ^E ^E ^E ^E ^X ^X ^X ^E ^E ^E ^E ^X ^X ^X ^X ^E ^E
      ^E ^E ^E ^E ^E ^X ^X ^X ^S 2 0 ^G ^Z ^Z ^X ^X ^X ^X 5 ^G ^G
(057) ^K ^D
(058) r
(059) ^K e
(060) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^D ^D 1 ^G
(061) ^K ^D
(062) r
(063) ^K e
(064) *^C ^E ^W ^R* ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 3 ^G ^X ^X
      ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^X ^D 1 0 ^G ^X ^X
      ^X ^X ^X ^X ^S
(065) ^K ^D
(066) r
(067) ^K e
(068) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^X ^X ^X ^D ^D ^D ^D ^D ^D ^X ^X ^E ^X ^X ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D / c o s ( x ) ^S ^S ^S ^S ^S ^S ^S ^S
      ^S ^S ^S ^S ^S ^S ^S ^S ^E ^E ^E ^E ^E ^E ^E ^E ^E ^E ^E ^E ^E ^E
      ^E ^E ^E ^X ^S ^S ^S ^S ^S ^S ^S ^S ^X ^E ^S
      r ^S ^G
(069) ^K ^D
```

(045) コンパイル

(046) 実行

(047) エディット・モードにする。

(048) const amp=20 →const amp=30
rowmax=10 →rowmax=20

(052) const amp=30 →const amp=10

(056) if (row mod 5)=0 then→if (row mod 20)=0 then
col:=round(10*sin(x))→col:=round(5*sin(x))

(060) if (row mod 20)=0 then→if (row mod 10)=0 then

(064) * *間はカーソル移動のミス
const amp=10→const amp=30
if (i mod 5) then line[i]:='|';→if (i mod 10) then line[i]:='|';

(068) sin(x)に/cos(x)を追加。const amp=30の3の前にrを入力、カーソルを移動しこれを消去。

```
(070) r
(071) ^K e
(072) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^F ^F ^F ^F
      ^F ^F ^F ^F ^F ^F ^F ^F ^F ^E ^E ^E ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^G ^G ^S ^S ^S ^S ^S ^S ^E ^S ^S ^G ^G ^G ^G
(073) ^K ^D
(074) r
(075) ^K ^D e
(076) ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^E ^E ^E ^E ^G ^G ^G 0 ^G ^D ^D 6 0 ^G ^G ^G ^X
      ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^S 0 ^G ^G ^G ^G ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^S 6 0 ^G ^G ^G
(077) ^K ^D
(078) r
(079) e
(080) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^D 0 ^G ^G ^G ^G
      ^F ^F ^F ^S ^S ^S ^S 6 0 ^G ^G ^G ^X ^E ^E ^E ^E ^E ^E ^E ^S ^S ^S
      ^S ^S 0 ^G ^G ^G ^G ^F ^F ^F ^S ^S ^S ^S 6 0 ^G ^G ^G ^E ^E ^E ^E
      ^E ^S ^S ^S ^S ^S ^S ^S ^S ^S ^S ^S ^S 0 ^G ^G ^G ^G ^D ^D ^D ^D
      6 0 ^G ^G ^G ^J
(081) ^K ^D
(082) r
(083) ^K e
(084) ^D ^D ^D ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^X ^D ^D ^D ^D
      ^D ^D ^D ^D ^D a m p ^G ^G ^X ^X ^X ^X ^X ^X ^S ^E ^S a m p ^G ^G
      ^X ^X ^X ^X ^S ^X ^X ^X ^S ^S ^S ^S ^S ^S ^S ^S 0 . 5 * ^X ^D ^X
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D a m p ^G ^G ^X ^X ^S ^S ^S ^S
      ^S ^S ^S ^S ^S ^S ^S a m p ^G ^G
(085) ^K ^D
(086) r r
(087) e
(088) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^D ^D ^D ^D ^D ^X ^X
      ^X ^X ^S ^X ^D ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D 1 0
      * ( ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D s^9 )
(089) ^K ^D
(090) r
(091) ^K ^D e
(092) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^X ^E ^D ^D ^D ^D
```

(072) col:=round(10*sin(x)/cos(x))→col:=round(sin(x)/cos(x))
x:=0.5*row;→x:=row;
(076) line:array[-amp..amp]of char;→line:array[6..60]of char;
if (col>=-amp)and(col<=amp)thenif→(col>=0)and(col<=60)then
(080) for i:=-amp to amp do write(line[i]);→ for i:=0 to 60 do write(line[i]);
for col:=-amp to amp do→for col:=0 to 60 do
for i:=-amp to amp do begin for i:=0 to 60 do begin

この時点でのプログラム

```
 1.        program sankakukansu;
 2.           const amp=30;
 3.                 rowmax=20;
 4.           var
 5.              line:array[0..60]of char;
 6.              row,col,i:integer;
 7.              x:=real;
 8.        begin
 9.           for row:=0 to rowmax do begin
10.              for i:=0 to 60 do begin
11.                 line[i]:=' ';
12.                 if (i mod 10) then line[i]:='|';
13.              end;
14.              if (row mod 10)=0 then
15.                 for col:=0 to 60 do
16.                    line[col]:='-';
17.              x:=row;
18.              col:=round(sin(x)/cos(x));
19.              if (col>=0)and(col<=60)then
20.                 line[col]:='*';
21.              for i:=0 to 60 do write(line[i]);
22.              writeln
23.           end
24.        end.
```

(084) line:array[0..60]of char;→line:array[0..amp]of char;
for i:=0 to 60 do begin→for i:=0 to amp do begin
x:=row;→x:=0.5*row;
if (col>=0)and(col<=60)then→if (col>=0)and(col<=max)then
for i:=0 to 60 do write(line[i]);→for i:=0 to amp do write(line[i]);
(086) 実行2回
(088) col:=round(sin(x)/cos(x));→col:=round(10*(sin(x)/cos(x)));
(091) ^K^Dは、エディット終了。ここでは不要な操作。
(092) カーソル移動が行なわれた。

```
      ^D ^Z
(093) ^K ^D
(094) r
(095) e
(096) ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^E ^E ^E ^D 0
(097) ^K ^D
(098) r
(099) ^K e
(100) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X
      ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^E ^E ^E ^E ^E ^E ^E ^E ^E
      ^E ^E ^E ^E ^E ^E ^E ^E ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X
      ^X ^X ^D ^E ^E ^E ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^G ^G ^G ^G ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^G
(101) ^K ^D
(102) r
(103) e
(104) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^E ^E ^E ^E ^E ^E ^E ^E ^E ^D 6 ^G
(105) ^K ^D
(106) r
(107) ^G e
(108) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^S ^X ^D ^D ^D ^D ^D 1 0 +
      BS ^G *
(109) ^K ^D
(110) r
(111) ^K ^D e
(112) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^X ^X ^X ^X
      ^X ^S ^S ^S ^S ^S 3 ^G
(113) ^K ^D
(114) r
(115) ^K e
(116) ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^X ^S ^X ^X ^X ^X ^X ^D ^D ^D ^D ^S ^E ^E ^D 5 ^G ^G
(117) ^K ^D
(118) r
(119) ^K e
```

(096)　x:=0.5*row;→x:=0.05*row;

(100)　カーソル上下移動。
col:=round(10*(sin(x)/cos(x)));→col:=round(sin(x)/cos(x));

(104)　const amp=30;→const amp=60;

(107)　^Gは不要な操作。

(108)　col:=round(sin(x)/cos(x));sinの前に10+と入力、+を消去し*を入力。
col:=round(10*sin(x)/cos(x));

(112)　col:=round(10*sin(x)/cos(x));→col:=round(30*sin(x)/cos(x));

(116)　if (i mod 10) then line[i]:='|';→if (i mod 5) then line[i]:='|';

```
(120) ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^D ^D ^D ^D ^E
      ^E ^E ^E ^E ^E ^E 8 ^G ^X ^X ^X ^X ^E ^E ^E ^D ^X ^X ^X ^X ^X ^X
      ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^X ^X ^S ^D ^S 5 ^G ^G
(121) ^K ^D
(122) r
(123) e
(124) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^W ^E ^E ^E ^E ^E ^E ^E ^E ^E
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 7 5 ^G ^G
(125) ^K ^D
(126) r
(127) ^K e
(128) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 6 0 ^G ^G
(129) ^K ^D
(130) r
(131) ^K e
(132) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X
      ^E ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^S ^S ^S 1 ^G
(133) ^K ^D
(134) r
(135) ^K e
(136) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^W ^W ^W ^W ^W ^E
      ^E ^E ^E ^E ^X 2 ^G
(137) ^K ^D
(138) r
(139) ^K e
(140) ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^S 1
(141) ^K ^D
(142) r
(143) ^K e
(144) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^E ^G ^E ^E ^E ^X ^X ^X ^X ^X
      ^S 5 ^G
(145) ^K ^D
(146) r
(147) ^K e
(148) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^S ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X
      ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D 0
```

(120) const amp=60;→const amp=80;
if (row mod 10)=0 then→if (row mod 5)=0 then

(124) const amp=80;→const amp=75;

(128) const amp=75;→const amp=60;

(132) if (row mod 5)=0 then→if (row mod 1)=0 then

(136) if (row mod 1)=0 then→if (row mod 2)=0 then

(140) if (i mod 5) then line[i]:='|';→if (i mod 15) then line[i]:='|';

(144) if (i mod 15) then line[i]:='|';→if (i mod 5) then line[i]:='|';
if (row mod 2)=0 then→if (row mod 5)=0 then

(148) x:=0.05*row;→x:=0.005*row;

```
(149) ^K ^D
(150) r
(151) ^K e
(152) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^G 1 ^G
(153) ^K ^D
(154) r
(155) ^K e
(156) ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^E ^E ^E ^D ^D ^D ^D 6 ^G r ^S ^G
(157) ^K ^D
(158) r
(159) ^K e
(160) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^X ^X ^X ^X ^D ^E ^E ^E ^E ^E ^E ^X ^X ^X ^X ^X ^X ^X ^X ^X
      ^X ^X ^D ^E ^E ^D ^D ^D ^D ^D ^D ^D ^S ^S ^S ^S ^S ^E 5 ^G
      ^X ^D ^D ^D ^D ^D ^D ^D ( ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D )
(161) ^K ^D
(162) r
(163) e
(164) ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^X ^X ^X ^X ^D ^D ^D 2 0 ^G ^S 5 ^G
(165) ^K ^D
(166) r
(167) e
(168) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^X ^D ^D ^D ^S 1 5 ^G ^G
(169) ^K ^D
(170) r
(171) ^K e
(172) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^F ^F ^F ^F ^F ^F ^F ^X ^X ^X ^X ^X
      ^D ^E ^E ^E ^D ^D ^E ^E ^D ^D 2 ^G
(173) ^K ^D
(174) r
(175) ^K e
(176) ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^S ^S
      1 5 ^G ^G
(177) ^K ^D
(178) r
```

(152)　x:=0.005*row;→x:=0.01*row;

(156)　x:=0.01*row;→x:=0.06*row;　修正中 rを入力したが、直ちに消去。

(160)　x:=0.06*row;→x:=0.05*row;
round(30*sin(x)/cos(x));→round(30*(sin(x)/cos(x)));

(164)　if (i mod 5) then line[i]:='|';　5を20に変更、さらに25にする。
if (i mod 25) then line[i]:='|';

(168)　if (i mod 25) then line[i]:='|';→if (i mod 15) then line[i]:='|';

(172)　if (i mod 15) then line[i]:='|';→if (i mod 12) then line[i]:='|';

(176)　rowmax=20;→rowmax=15;

```
(179) e
(180) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^D ^D 5 ^G ^K ^S
(181) ^K ^D
(182) r
(183) ^K e
(184) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^E ^S
      2 ^G ^S ^S ^S ^S ^S ^S
(185) ^K ^D
(186) r
(187) e
(188) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^S ^S
      4 ^G
(189) ^K ^D
(190) r
(191) ^K e
(192) ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D s ^S
      ^S ^S 2 5 ^G ^G ^G
(193) ^K ^D
(194) r r
(195) e
(196) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^D ^D 0 ^G
(197) ^K ^D
(198) r
(199) e
(200) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X 3 ^G ^J
(201) ^K ^D
(202) r
(203) ^E e
(204) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^X 1 5 ^G ^G ^S ^S ^S ^S ^E ^S
(205) ^K ^D
(206) r
(207) ^K e
(208) ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^S ^S 5 ^G
(209) ^K ^D
```

(180) round(30*sin(x)/cos(x));→round(50*sin(x)/cos(x));

(184) round(50*sin(x)/cos(x));→round(20*sin(x)/cos(x));

この時点でのプログラム

```
 1.      program sankakukansu;
 2.        const amp=60;
 3.              rowmax=15;
 4.        var
 5.          line:array[0..amp]of char;
 6.          row,col,i:integer;
 7.          x:=real;
 8.      begin
 9.        for row:=0 to rowmax do begin
10.          for i:=0 to amp do begin
11.            line[i]:=' ';
12.            if (i mod 12) then line[i]:='|';
13.          end;
14.          if (row mod 5) 0 then
15.            for col: 0 to 60 do
16.              line[col]:='-';
17.            x: 0.05*row;
18.            col:=round(20*(sin(x)cos(x)));
19.            if (col> 0)and(col< amp)then
20.              line[col]: '*';
21.            for i: 0 to amp do write(line[i]);
22.            writeln
23.          end
24.        end.
```

(188) rowmax 15;→rowmax 45;

(192) rowmax 45;の後にsを入力、カーソル左へ、45の前に25を入力、45sを消去。
rowmax 25;とする。

(196) rowmax 25;→rowmax 20;

(200) col: round(20*(sin(x)cos(x)));→col:=round(30*(sin(x)cos(x)));

(204) rowmax 20;→rowmax=15;

(208) if (i mod 12) then line[i]: '|';→if (i mod 15) then line[i]: '|';

```
(210) r
(211) e
(212) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 5 ^G
(213) ^K ^D
(214) r
(215) ^K e
(216) ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^X ^X ^X ^X
      ^X ^X ^X ^X ^X ^E ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^S ^S ^G ^G ^G
(217) ^K ^D
(218) r
(219) e
(220) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D 1 ^G
(221) ^K ^D
(222) r
(223) e
(224) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^E
      ^E ^S 1 0 * ^S ^S ^S ^S ^S ^S ^S ^S ^E ^S ^G
(225) ^K ^D
(226) r
(227) e
(228) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^S 6 ^G
(229) ^K ^D
(230) r
(231) e
(232) r ^S ^G ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^
      D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^S 2 ^G s ^S ^S 1 5 ^G
      ^G
(233) ^K ^D
(234) r
(235) e
(236) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X 0 9 ^G ^G
(237) ^K ^D
(238) r
```

(212)　col:=round(30*(sin(x)cos(x)));→col:=round(50*(sin(x)cos(x)));

(216)　col:=round(50*(sin(x)cos(x)));→col:=round((sin(x)cos(x)));

(220)　x:=0.05*row;→x:=0.01*row;

(224)　col:=round((sin(x)cos(x)));→col:=round(10*(sin(x)cos(x)));
　　x:=0.01*row;→x:=0.1*row;

(228)　if (i mod 15) then line[i]:='|';→if (i mod 16) then line[i]:='|';

(232)　文頭にrを入力、これを消去。x:=0.1*row;をx:=0.2*row;としたが、sを入り、
　　さらに、定数を0.15にした。x:=0.15*row;

(236)　x:=0.15*row;→x:=0.09*row;

(239) e
(240) x x x x x x x x x x ^S ^S ^S ^S ^S ^S ^S ^S ^S ^S ^G ^G ^G ^G ^G
^G ^G ^G ^G ^G ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^E ^E ^E ^E ^D
^D ^D ^D 3 ^G
(241) ^K ^D
(242) r
(243) e
(244) r ^S ^G
(245) ^K ^D
(246) r
(247) e
(248) ^C ^W ^E ^E ^E ^W ^Q ^K ^X ^X ^X ^X ^X ^X ^S ^X ^X ^X ^X ^X ^X ^X
^D ^D ^E ^E ^D ^D ^D ^D 2 ^G ^G
(249) ^K ^D
(250) r
(251) ^K e
(252) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^E ^E ^E ^D ^X ^X ^S 1 0 ^
(253) ^K ^D
(254) r
(255) ^K e
(256) ^C ^R ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^X ^X ^X ^X ^D ^D ^D 2 0 ^G ^X ^X ^S ^S ^S d
^F ^S ^S ^S ^S ^S ^G ^D ^D ^D ^D ^G
(257) ^K ^D
(258) r
(259) ^K ^D e
(260) ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^D ^X
^D ^X ^D ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^S ^S 5 ^G
^G ^X ^X ^S ^S 5 ^G ^G
(261) ^K ^D
(262) r
(263) ^K e
(264) ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^X ^X ^X ^X ^D ^E ^D ^D ^D ^D ^D 6 0 ^G
(265) ^K ^D
(266) r
(267) e

(240) 文頭にxを10個入力、これを消去。コントロールキーの押し忘れと考えられる。
if (i mod 16) then line[i]:='|';→if (i mod 13) then line[i]:='|';

(244) 文頭にrを入力、これを消去。エディット終了し、再実行((246)まで)。

(248) if (i mod 13) then line[i]:='|';→if (i mod 2) then line[i]:='|';

(252) if (row mod 5)=0 then→if (row mod 10)=0 then

(256) if (i mod 2) then line[i]:='|';→if (i mod 20) then line[i]:='|';

(260) if (i mod 20) then line[i]:='|';→if (i mod 5) then line[i]:='|';
if (row mod 10)=0 then→if (row mod 5)=0 then

(264) if (i mod 5) then line[i]:='|';→if (i mod 60) then line[i]:='|';

(268) ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^X ^X ^X ^X ^X ^X ^X ^X ^X ^E ^E ^E ^D
^D ^D 4 ^G
(269) ^K ^D
(270) r
(271) e
(272) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^W ^W ^W ^W ^E ^E 3 ^G
(273) ^K ^D
(274) r
(275) e
(276) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X 5 ^G ^G
(277) ^K ^D
(278) r
(279) e
(280) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 4 ^G
(281) ^K ^D
(282) r
(283) e
(284) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^D 1 0 ^G
(285) ^K ^D
(286) r
(287) e
(288) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^S 5 ^G ^G
(289) ^K ^D
(290) r
(291) ^K e
(292) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^S 6 ^G
(293) ^K ^D
(294) r
(295) ^K e
(296) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^W ^W ^A ^W ^E ^D ^D
^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 2 0 ^G

```
(268)  if (row mod 5)=0 then→if (row mod 4)=0 then

(272)  if (row mod 4)=0 then→if (row mod 3)=0 then

(276)  if (i mod 60) then line[i]:='|';→if (i mod 5) then line[i]:='|';

(280)  if (i mod 5) then line[i]:='|';→if (i mod 4) then line[i]:='|';

(284)  if (row mod 3)=0 then→if (row mod 10)=0 then

(288)  if (row mod 10)=0 then→if (row mod 5)=0 then

(292)  if (row mod 5)=0 then→if (row mod 6)=0 then

(296)  if (i mod 4) then line[i]:='|';→if (i mod 20) then line[i]:='|';
```

```
(297) ^K ^D
(298) r
(299) ^K e
(300) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^D 4 ^G
(301) ^K ^D
(302) r
(303) e
(304) ^X ^X ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^S 1 5 ^G
(305) ^K ^D
(306) r
(307) ^K e
(308) ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X ^X ^X ^S ^E ^E ^E ^E ^E
      ^E ^E ^E ^E ^E ^S ^S ^S ^S ^S ^S ^S ^S ^S 4 0 ^G ^G ^X
      ^X ^X ^D ^D ^D ^X ^X ^X ^X ^D ^D ^X ^X ^X ^D ^D ^D ^S 4 ^G
(309) ^K ^D
(310) r
(311) e
(312) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^S 6 ^G ^X ^X ^X ^X
      ^X ^X ^X ^X ^X ^D ^X ^D ^D ^D ^D ^D ^D ^D ^D ^S 2 5 ^G ^G
(313) ^K ^D
(314) r
(315) ^K e
(316) ^X ^X ^X ^X ^X ^X ^X ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D
      ^D ^D ^D ^D ^D ^D ^D ^D ^X ^X ^X 5 0 ^G ^G
(317) ^K ^D
(318) r
(319) ^K e
(320) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 8 ^G
(321) ^K ^D
(322) r
(323) e
(324) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 7 ^G
(325) ^K ^D
(326) r
(327) ^K e
(328) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 4 ^G
```

```
(300)  if (row mod 6)=0 then→if (row mod 4)=0 then

(304)  if (row mod 4)=0 then→if (row mod 15)=0 then

(308)  const amp=60;→const amp=40;
       if (i mod 20) then line[i]:='|';→if (i mod 40) then line[i]:='|';

(312)  const amp 40;→const amp 60;
       if (i mod 40) then line[i]: '|';→if (i mod 25) then line[i]: '|';

(316)  if (i mod 25) then line[i]:='|';→if (i mod 50) then line[i]:='|';

(320)  const amp 60;→const amp 80;

(324)  const amp 80;→const amp 70;

(328)  const amp 70;→const amp 40;
```

```
(329) ^K ^D
(330) r
(331) e
(332) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 5
(333) ^K ^D
(334) r
(335) e
(336) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^S ^S ^G
(337) ^K ^D
(338) r
(339) ^K ^D e
(340) ^X ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D ^D 6 5 ^G ^G
(341) ^K ^D
(342) r
(343) e ^K ^E ^K ^D
(344) r r
(345) s
(346) q
```

(332) const amp=40;→const amp=540;

(336) const amp=540;→const amp=50;

(340) const amp=50;→const amp=65;

(345) 保存

(346) 終了

研究発表－1

社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
ET94-65 (1994-09)

Logoプログラム作成過程の分析

近藤智嗣

新学社

〒162　新宿区市谷山伏町1-19

中野靖夫

上越教育大学

〒943 上越市西城町1-7-2

あらまし　中学校の情報基礎におけるプログラミングの学習過程の実態を明らかにするためにキー入力データを記録するプログラムを使用し，Logo言語の学習履歴を分析した。Logo言語の学習が10時間目の授業のデータを分析したところ，プログラム例を参考に入力するという課題では，ミスの多くは実行する前の入力時に修正されている。実行時のエラーの発生箇所は組み込み手続きよりもユーザーの定義した手続き名に起因する場合が多く，かつ誤りに気づきにくい。またプログラム内にカナ入力を併用するとタイピングミスが増加する等の知見が得られた。今後さらに各学習段階での分析を進めることで情報基礎におけるプログラミング指導の適切な一手法を開発できると考える。

和文キーワード　Logo，プログラミング，操作過程，中学生

An analysis of programming process in the language Logo

Tomotsugu KONDO

Shingakusha Co.,LTD.

1-19 Ichigaya Yamabushi-cho
Shinjuku-ku Tokyo 162 Japan

Yasuo NAKANO

Center for Educational
Research and Development
Joetsu University
1-7-2 Nishisiro,Jyouetu-shi
Niigata 943,Japan

Abstract　To make clear the actual situation on the process of learning the programing in the information study at junior high school,We analized the learning process in Logo language through using the program which records the user data of key operation.Analyzing the data in the tenth class of Logo language learning,the subject which refers to the sample program,most of the user's mistakes are corrected before running the program.Error breaking out points are due to the procedure name which user defines rather than primitive procedure,and hard to notice the mistakes.And We noticed that in the program,using the kana operation at the same time makes user more mistakes.After this,analyzing the each learning step more,We will be able to find the suitable means of programming leading in imformation study.

英文 key words　Logo,Programming,Operation Process,Junior High School Student

1．はじめに

平成5年度からの中学校学習指導要領の実施に伴い，技術・家庭科の1領域として情報基礎が設けられた。この指導要領によると情報基礎の目標は「コンピュータの操作を通して，その役割と機能について理解させ，情報を適切に活用する基礎的な能力を養う。」と掲げられている。この目標に到達するため，授業では応用ソフトとしてワープロ，表計算，グラフックス，データベースが使用されている。また，指導要領の内容には「プログラムの機能を知り，簡単なプログラムの作成ができること。」とあり，授業においてもプログラミングが徐々に導入されるようになってきている。プログラミングの使用言語には教育用としてLogoとBASICが多く利用されている。

しかし，実際の情報基礎の授業は，コンピュータに慣れることに重点が置かれ，グラフィックスソフトで絵を描き，ワードプロセッサで文書作成するだけの場合が多く，これらのソフトウェアを使用した成果としてプリンタから出力された成果物によってのみ評価されているのではないだろうか。プログラミングの場合も同様であり，例題通り入力し動作を確認するという形式が多いようである。これは，情報基礎の授業が始まって間もないため学習者の学習過程が指導者に把握されておらず，その指導法が確立されていないためだと思われる。

学習における指導方法においてはその結果である成果物からのみの評価ではなく，その作成過程も考慮にいれなければならない。また，その作成過程における学習者のつまずき，誤り等を明らかにすることはその指導法を確立する意味からも重要である。

これまで，中学生における知識生成としてのプログラム作成過程の分析は，あまり分析されていない。プログラミングのように確実に手順を追わなければならない学習では一度誤りが生じるとその先に進むのは難しいため，誤りの原因を追求することも必要である。

そこで，本研究ではプログラミングの作成過程，誤りの発生しやすい箇所等を分析し，また，学習者の理解の状況を明らかにすることにした。

2．方法

学習にあたり本時の課題とプログラム例，必要な命令の解説を記した資料を配布し，課題を作成させた。そしてそのキー操作履歴データを分析した。

2-1 被験者

山形県内の公立中学校 2年生24名

2-2 実施日

1994年2月26日

「選択技術」の時間に実施した。

2-3 学習形態

被験者は24名であるが，2名1組で1台のパソコンを使用したものもあり，データ数は17である。

2-4 学習経験

Logo学習は10時間目

2-5 使用機器

・NEC PC-9801

・LogoWriter2（ロゴジャパン）

・キー操作の再現システム

2-6 課題

[課題1]以下の2つのプログラム例（図1）を参考にして，タートルを一方向に動かす手続きを作成する。

```
（タートルの色，形，位置を決める）
  TO KAME
  SETC 3 SETSH 2
  PU
  SETPOS [50 50]
  SETH 90
  END
-----------------------------------------
（タートルを動かすプログラム）
  TO UGOKU 1
  REPEAT 50 [FD 3]
  END
```

図1 課題1の完成プログラム例

[課題2]

動きの速さを変える命令を適切な場所に追加する

（命令は配布した資料に記入されている）

[課題3]
タートルの色，形を変えないでいろいろな動きをさせる手続きに加工する
[課題4]タートルの色，形を変えていろいろな動きをさせる手続きに加工する
（課題3～4については，自由に生徒に作成させた。）

2-7 データ収集と分析法

前田ほか(1993)によって開発されたシステムを使用してキー操作履歴を収集した。そして，そのデータをエディタを使用し，リターンキーと画面を切り換える[f･6][f･7]キーのところで改行した。こうすることによって1行単位の見やすいデータとなり，これを印刷したものを用意する。次にキー操作履歴の再現システムを使用し，被験者が操作した通りに再現する。この画面上での状態を見ながら印刷したデータ用紙に操作行動のカテゴリー，エラーの発生箇所，状態等を記述する。

3．結果と考察

3-1 課題の達成度

課題1は17名で全員，課題2は14名，課題3は12名，課題4は4名が達成した。

3-2 課題1のカテゴリー分析

ここでは，全員が到達した課題1に関してカテゴリー分析した。表1がそのカテゴリーと結果である。

課題1はプログラム例を参考にして手続きを作成するという課題であり，編集画面にプログラムを入力し，実行した結果，不具合があれば修正する。そしてさらに加工しながら完成させていくという過程をとる。この過程の中では，プログラムの構造を考える論理的なことと，タイピングなどの技能的なことが必要になる。その二つの要因ごとに分析しなければならない。

表1 課題1のカテゴリーとその結果

			誤り箇所	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	計
入力	正	修正なし		5	6	5	2	7	1	4	6	3	13	5	5	7	6	2	9	6	92
		入力中修正	書式	1	1					1		1	1		1		1	1			8
			命令			1	2	3	5	2	1	4	1		3	1	1	2	1	1	28
			数	1		1	3		2	3	1		1	3	1			1		1	18
			手続き名				2		1	2					1	1			1		8
			空白挿入	3								2	1					1			7
	誤		書式													1	1				2
			命令			1			1								1				3
			数																		0
			手続き名																		0
			空白挿入		1				1	2				1				1			6
	不要な入力							2						1							3
修正	正	実行後修正	書式																		0
			命令			1				2											3
			数																		0
			手続き名																		0
			空白挿入							1				1							2
			空白削除		1				2									1			4
	誤		書式																		0
			命令																		0
			数																		0
			手続き名																		0
			空白挿入																		0
			空白削除							2											2
	不要な修正				1	2			6				3								12
加工	正		数値変更							2											2
			命令変更					1					2								3
			命令追加																		0

まず，カテゴリーは入力，修正，加工に分け，それらの正誤，誤りの箇所で分類した。入力は結果として論理的に正しい場合でも，そのまま入力した場合と，途中で修正しながら入力した場合に分けた。修正は実行後に誤りを直した場合である。

結果をみると間違いの多くは入力途中で誤りに気づき修正されていることがわかる。中でも，命令のスペルミス，引数となる数を変更しているものが多い。また，命令と命令，命令と引数の間には空白を入れて区切りとするが，空白を入れ忘れるミスが7件あった。

逆に空白を入れてはいけないところに空白を入れてしまったミスは6件あり，これは実行後エラーが発生してから修正されている。その他には，見かけ上は正しく見えても，前の行で改行されておらず2行がつながってしまってエラーとなったものもいた。命令のミスよりも空白の有無によるミスなどの方が初学者にとっては気づきにくいと言えるだろう。

また，誤りではないが，行や文字を挿入する場合，挿入する前におおよその文字数分空白や改行を入れたり，正しく入力されている命令を一旦消し，挿入後再入力するという例も少なくなかった。エディターの挿入の概念が把握されていないための不要な動作である。

3-3 課題2の結果

課題2はタートルの動く速度を変える命令SLOWTURTLE，FASTTURTLEを適切な位置に追加するという課題である。これができたのは14名であった。挿入する位置は既に作成してある2つの手続きの中に追加するのだが，結果は，コマンドラインに入力した1名を除いて13名がタートルを移動させる手続きの方に挿入していた。その手続きの中での位置は図2に示すように3箇所に分けることができる。①の位置が4名，②の位置が2名，③の位置が7名であった。①と②は実行結果としては同じになるので，合計すると6名で約半数がタートルを移動させる前に，残りの半数がタートルを移動させた後に設定したことになる。③の位置は文法的には誤りではないが，実行時には有効でない位置である。再度実行すると前の設定が保たれているので，結果もタートルの動作速度が変化する。この被験者群は，それぞれの手続きの役割は理解しているが各行の実行順序までは理解が及んでいないと言える。

```
TO UGOKU1
①
② REPEAT 50 [FD 3]
③
END
```

図2 SLOWTURTLEの挿入位置

3-4 実行時のエラーの分析

編集画面でプログラムが一応完成すると実行画面に切り換えて実行する。その実行画面のコマンドラインでのエラーを分析した。

エラーが生じるとエラーメッセージかシステムより返される。特に頻度の多かったエラーメッセージを以下に示す。

```
①かめのしかたは知りません
②てじゅんはのインプットがおかしいです
③qの中でFD3のしかたは知りません
④!の中でREPEATのｲﾝﾌﾟｯﾄがおかしいです
⑤らの中で[]をどうするのかわかりません
```

図3 エラーメッセージの例

そこで，1時間の授業の中でのエラーメッセージを原因別に分類したのが表2である。

数値の上段はエラーメッセージが表示された回数，すなわち誤ったものを実行した回数である。下段はプログラム上，またはコマンドライン上での誤りの個数である。従って，例として被験者2の誤った手続き名の場合は，1つの誤りに対して9回実行を繰り返したということになる。

ここで言う命令とはLogoシステムにあらかじめ組み込まれている手続きであるが，このスペルを間違えたのは1件，別の命令を使用してしまったのが1件となっており，出現件数は少ない。両者とも間違いをすぐに発見し，1回で修正している。

手続き名とはプログラム作成者が定義する手続きの名前であるが，これに関してのエラー出現件数は多く，誤った手続き名を入力したのが9件，それを繰り返して実行しエラーの発生は26件に及んでいる。また，手続き名の中に空白を入れてはならないが，UGOKU 1のように途中に空白を入れてしまったのは4名で，誤りの箇所は

表2 エラーの分類

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	合計
命令のｽﾍﾟﾙﾐｽ			1															1
			1															1
誤った命令の使用				1														1
				1														1
誤った手続き名		9		2		9	1				2		3					26
		1		2		2	1				2		1					9
手続き名中に空白		2					4				10				3			19
		1					2				1				1			5
引数忘れ				1	1													2
				1	1													2
空白忘れ	1			1								1					1	4
	1			1								1					1	4
改行忘れ						6												6
						1												1
括弧忘れ									1		1							2
									1		1							2
構文の間違い									2				1	1				4
									1				1	1				3
引数が範囲外								1								2		3
								1								1		2
入力ミス	1					1				2								4
	1					1				2								4
ファイル名に空白							2											2
							1											1
ﾌｧｲﾙ名が不適切						1	1											2
						1	1											2
その他	1									3								4
	1									3								4

5件，エラーの発生件数は19件となっている。この原因は，配布した資料に見かけ上若干空きがあるように見えるため，その影響が現れたと思われる。

命令の誤りが少なかったのは，配布した資料とディスプレイを見比べているため，入力の段階で既に修正を施すことができたと思われる。また，手続きの誤りが多かったのは，手続き名をプログラム作成者自身で定義しており自由度が高いためと思われる。誤ったまま何度も実行を繰り返しているためエラーの件数も多い，つまり，気づきにくい誤りと言えるだろう。こうした手続きも学習の際にはある程度決めた言葉を使うことで誤りの原因を少しでも排除できるのではなかろうか。

その他，引数，括弧の忘れは少なく2件であった。空白の忘れは4件あり，これは，命令と命令，命令と引数の間に空白を入れなかったものだが，表1の入力途中での修正は7件であり，この時点で修正し忘れたものであろう。

3-5 キー入力ミスの分析

もう一つの観点である，タイピングの技能面として，1時間の授業中のキー入力ミスを分類したのが表3である。

表3 キー入力の操作ミス

ｶﾃｺﾞﾘｰ	内容	数
ｼﾌﾄﾐｽ	ｼﾌﾄｷｰを押さなかった	7
	不要なｼﾌﾄｷｰを押した	1
ｶﾅﾐｽ	ｶﾅ入力でｶﾅｷｰを押さなかった	10
	不要なｶﾅｷｰを押したまま	28
類似ｷｰ	形の似たｷｰ	24
隣接ｷｰ	隣のｷｰ	11

使用したLogo言語は大文字小文字の区別がないため，SHIFTｷｰは記号の入力時にのみ必要となる。ここで必要な記号は"，:，[，]である。そこで，記号の入力ミスをみると，"の入力でｼﾌﾄｷｰを押さなかったのが7件あった。

また，ファイル名や手続き名にカナを使用し

ているのは17名中14名で，その内，何らかのミスをしたものは13名とカナを使用したほとんどのものがｶﾅｷｰの操作時にタイプミスをしていた。ｶﾅｷｰに関係するミスの総数は38件に及んでいた。

類似キーとは，iとl，LとT，ソとン，.と,等の間違いのことである。

3-6 被験者別操作特性

操作過程は被験者ごとに特性がある。ここでは，特徴的な操作過程のある被験者について考察する。

[被験者1]

区切り文字としての空白を忘れたのが9箇所，ほとんどはすぐに気づき直している。

命令を挿入する際，あらかじめ命令の文字数分空白を挿入しておきそれから入力する。挿入した空白は削除している。

命令の一部分を加工する際（slowturtle → fastturtle），fastをt←s←a←fのように逆から1文字ずつ挿入している（fastturtle → slowturtle）の場合も同じ。

これらのことから，エディタの挿入の概念が十分把握されてないと思われる。

カナキーを押さずにカナを入力し，カナキーをはずさずにアルファベットを入力するというミスが3箇所あった。iとlの押し間違いが2箇所。.(ﾋﾟﾘｵﾄﾞ)の入力で，/を4回，?を2回，>を1回押した後入力できた。

[被験者2]

手続き名に空白を入れてしまったためエラーが生じ，そのまま，ｶﾒと入力したが手続き名はローマ字でKAMEであるので別のエラーが生じた。同時に2つのエラーが発生した状態となり，手続き名の違いは9回のエラー発生の後修正。手続き名に空白を入れた誤りはその途中で2回のエラー発生後修正。結局，修正に手間取り課題は1までしかできなかった。

エラーが発生した後，一旦編集画面に戻すが，何も手を付けず再度実行する場合が多い。エラーが出たときの対処のしかたがわからないのであろう。

[被験者6]

全角カナで8文字のファイル名を付けてエラー。

TO文でヘリという名の手続きを定義した後，TABｷｰを数回押して行をまたがりそのまま次の行を入力したため，1行目が改行されておらずエラーとなった。これに対して「てじゅんはのインプットがおかしいです」「ヘリのしかたは知りません」が数回表示されたが直らないため，手続き名のまったく異なるKAMEという命令を入れて動かそうとしてまたエラーが出た。これに起因するエラーの合計数は15回であった。

カナキーの操作による入力ミスは8回，その他の入力ミスは4回であった。この被験者もエラーの修復に手間取り課題は2までしかできなかった。

4．おわりに

プログラミングの作成過程，誤りの箇所，修正の方法等を分析することにより，学習者の理解状況を把握することができる。それにより，指導者は学習者に対して適切なフィードバックを与えることができるようになる。また，提示する課題についても誤りの原因となる不要な情報を排除し効率の良い学習が可能となる。さらに使用するシステムについても学習の妨げになる機能の改善点等を獲得できる考えられる。

本研究では，一般的な授業のデータを使用し，被験者数も少なかったが，今後は実験計画のもとにデータを収集し，情報基礎におけるプログラミング指導の一方法を確立していくことが課題である。

なお本研究は文部省科学研究費補助金，一般研究C（05680172）"プログラム作成過程に視点をおいた評価法の開発"による研究成果の一部である。

参考文献

1)前田恵三，中野靖夫:コンピュータ操作過程の再現システム，日本教育工学雑誌，Vol.16 No.4，pp.185-195,1993

2)近藤智嗣，中野靖夫:Logo学習におけるキー操作の分析，日本教育工学会第9回大会講演論文集pp.310-311,1993

研究発表－２

社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
ET94-66 (1994-09)

C言語のプログラミング過程の分析

前田　恵三
東京工業高等専門学校
情報工学科

〒193　東京都八王子市椚田町 1220-2
Phone: 0426-68-5064
E-mail: keizo@cs.tokyo-ct.ac.jp

中野　靖夫
上越教育大学
学校教育研究センター

〒943　新潟県上越市西城町 1-7-2
Phone: 0255-25-9147

あらまし　パーソナルコンピュータを使ってC言語のプログラミング演習を行なっているときのキーの入力状況を記録し、プログラム作成過程の分析を行なった。記録したデータを整理したプリントアウトと操作の再現の観察によって作成過程を把握することができた。また、操作の状態をカテゴリーに分け、分析することで、各学習者のプログラミングの方法や傾向を把握できることが分かった。このプログラミング過程の分析によって、学生の理解度、苦手とする事項が把握でき、演習内容の適切さの判断や学生へのきめの細かい指導に役立てられると考える。

キーワード　プログラミング教育，学習支援，C言語

Analysis of Programming Process in C

Keizo MAEDA
Tokyo National College of Technology
1220-2 Kunugida, Hachioji, Tokyo 193, Japan
Phone: 0426-68-5064
E-mail: keizo@cs.tokyo-ct.ac.jp

Yasuo NAKANO
Joetsu University of Education
1-7-2 Nisisiro, Jyoetu, Niigata 943, Japan
Phone: 0255-25-9147

Abstract　We analyzed programming process of students by recording keyboard input history on exercise of the programming language C. Programming process is understandable by tracing computer operations and category analysis of operation state. And we think that results of our analysis are usable to support learning programming.

keywords　programming education, learning support, C language

1　はじめに

プログラミング言語の教え方としては、(1)文法と機能の説明、(2) プログラム例を手本として示す、(3) 例題の書き替え、(4) 課題の演習、の順に説明と演習を交互に行なうのが一般的である。そして、教師は提出されたレポートや試験の結果で理解の度合いを見たり、その学生の苦手なところや悪い癖を見つけ、指導を行なったりする。

東京高専情報工学科では入学時から３年間に渡ってＣ言語を用いたプログラミング教育を行なっているが、時間をかけた割には実用規模のプログラムを作れる学生が少なく、苦手意識を持っている学生も見受けられる。そのため、プログラミング言語の授業の成績に関係なく、プログラミングに自信のない４学年の学生を対象に、さらに１年間プログラミング演習を行なっている。

この演習を始めた最初の２年間は、プログラミング過程を見るために、パーソナルコンピュータを使わずに紙の上にプログラムを書かせ、進み具合を観察しながら個別に指導を行なった。そして、授業の終わりに、担当の学生に作成したプログラムについて黒板を用いて説明を行なわせた。

紙の上でプログラミングを行なわせたのは、キーボードを前にすると、うまくいかないときに、よく考えずにプログラムに変更を加える傾向があるのを感じていたためである。

紙の上でプログラムを行なわせたクラスは、教科書にカーニハン，リッチーの「プログラミング言語Ｃ」[1] を用いて３年間学習してきていた。そのためか、関数 scanf, printf を使った実用的なプログラムが苦手で、getchar などを使った文字列処理が得意であった。しかし、配列に入れてある文字列の末尾を EOF で見つけようとするなど、理解の浅い学生も何人かいた。

全体として、(1) ループの見つけ方（制御構造部の発見）、(2) ループの適切な使い方、for, while, do-while の選択、(3) 使うべき関数の判断に弱く、また、(4) フラグ変数を多用したり、(5) 途中までできていても、完成しないと消してしまう学生もいた。

紙の上でのプログラミングは、その場で学生に助言を行なって効果があるように思えたが、次の演習でも同様な手法の誤りを犯すことも多く、もう少し丁寧なデータ収集と分析の必要を感じた。

また、紙にプログラムを書いてから説明もさせるため、時間的な制約から小規模の内容の演習にとどまってしまった。

平成６年度は、パーソナルコンピュータを使用し、問題の内容も、制御構造の使い方の習得と、アルゴリズムの発見と実現に重点をおいた課題を行なうことにした。

ところで、プログラム作成過程で学生が何につまずき何を試そうとしたのか等は、提出された結果からは知ることはできない。それらはプログラミング過程を分析することで把握できると考えた。

本研究においては、コンピュータの操作過程を記録・再現するシステム[2] を使ってプログラミング過程の分析を行なった。

これまでの演習の操作記録の分析から、学生のプログラミングの進め方と、記録された操作過程に関連性がみられた。また、それを学生への指導に役立てられると考えている。

2　方法

2．1　被験者と課題内容

プログラミング過程のデータを収集する対象となる被験者は、Ｃ言語については十分教育を受けていて、パーソナルコンピュータの操作にも慣れている学生10名である。４月からの毎週90分の授業で課題を１から２個与え、時間内で解かせている。終わらない学生は自主的に授業時間を越えて演習を続け、30分を超過しない範囲で完成または動作するところまでにはなっている。

今回の報告では、以下の２つの課題についてデータの収集・分析を行なった。課題１は２次元配列と文字列の取り扱いがキーポイントになる。関数 strcpy や strcmp の使い方は TURBO-C のヘルプ機能を使って調べられる。課題２は多桁計算という学生にとって未知の分野であり、その数値を表現する手法、アルゴリズムの検討に時間がかかると予想した。加算を行なう２つの数値の桁数が異なるときと、元の値より桁数が増えるときの処理の必要性に気付いて、どう対処するのかも興味のあるところであった。

課題１．　テキストファイルの行単位のソート

文字列ファイルの各行を配列に読み込み、アルファベット順でソートして表示するプログラムを作成せよ。処理できる最大行数、1行の最大文字数は、200行、100文字とする。文字列では代入文が

使えないので関数 strcpy を使う。また、比較には strcmp を使えばよい。これらの関数を使うにはヘッダファイル string.h が必要である。入力ファイルはこのプログラムファイルでよい。

課題２．　多桁計算

int 型の配列の各要素に 10 進数の値の各 1 桁を入れて整数値を表わすとすると、配列の大きさ分の桁数を表現できることになる。

配列の要素 0 に現在入っている値の桁数 n を入れ、要素 1 に 1 桁目の値を、要素 n に n 桁目の値を入れておく。

値を表示するには、

```
int bignum[200];
for (i=bignum[0]; i>0; i--)
   printf("%d", bignum[i]);
```

値のセットには、文字列を使うことにする。関数 set は次のように書ける。

```
void set(int x[ ], char s[ ])
{
   int i;
   for (i=strlen(s)-1, j=1; i>=0; i--, j++)
      x[j]=s[i]-'0';
   x[0]=strlen(s);
}
```

使うときは、set(a,"685064"); のようにする。

２つの変数の加算は、下の桁の要素から加えていき、繰り上がり（carry）があれば、次の桁に送る。carry は（加算結果／ 10）で求められる。

演習

２つの変数の加算を行なう関数 add とテスト用のプログラムを作成せよ。add のプロトタイプ宣言は次の通りとする。

```
void add(int a[ ], int b[ ], int c[ ]); /* c = a + b */
```

■

なお、被験者の中には、紙にプログラムを書いてからキー入力する学生は一人もいなかった。また、フローチャートやPADなどのプログラム図式も一切使用していない。

使用した機材やＣ言語の処理系は次の通りである。

・パーソナルコンピュータ
　　日本電気（株）PC-9801 DS5
・Ｃ言語　BORLAND　TURBO C 2.0

２．２　データの収集と分析方法

操作過程を記録・再現するシステムを用いて、パーソナルコンピュータへのキー入力を、その前のキー入力からの経過時間と共に記録し、演習終了後に自動的にフロッピーディスクに保存した。分析手順は、

(1) 各学生が作成したプログラムリストと記録した操作過程のデータをプリントアウトし、プログラミング過程のシナリオを想像してプリントアウトした紙にコメントを付加していく。

(2) 操作過程をパーソナルコンピュータ上で再現し、操作過程のプリントアウトとともに操作状況を観察し、プリントアウトした紙のコメントを確かなものにする。

長い文章をメモするときには、再現を一時停止したり、学生が思考中でキー入力が無いときは再現を速めたりした。再現画面には経過時間が表示されているので、再現を速めても、思考したり悩んだ時間を見失うことはない。

この分析方法を使用すると、分析者にプログラミング経験があれば、学生一人一人のプログラミング過程を十分把握することが可能である。カーソル移動一つにしても、どのような意味があったのか、想像することができる。

操作過程のプリントアウトの一例を図１に示す。カーソルの移動ばかりしているように見えるが、丁寧に見ていくと、変数名を入力しようとして宣言していないことに気付き、頻繁に変数宣言を記述しに前の行へ戻っていることなどが分かる。

90分の操作記録を分析するのに、プリントアウトと再現の両方を利用すると45～60分を費やすが、最終結果には現われない学生の思い違い等を発見でき、得られる成果が大きい。

```
@f･10f･10▼289.1
▼290.0
〈〈〈b:▼292.5
↓ESC↓↓↓↓▼299.9
b:no-10▼306.6
@include□←←←←←←←←#→→→→→→→→→←<stdio.h>←←←←←←←←←←□
→→→→→→→→→→→→←←▼670.2
main()▼678.8
{▼680.4
□□int□bid〈gm〈〈gnum[200];▼697.3
for□(i=bignum[0];□i>0;□i--)▼718.7
□□□□printf("%d",bignum[i]);▼741.1
↑↑↑↑←↑←←←←←▼753.3
↑DEL↓↓↓↓↓↓↓}▼768.2
void□set(int□x[],□char□s[])▼785.2
{▼787.7
□□□〈□□int□i;▼796.0
for□(i=stren(s)〈〈〈(s)-1,□j=1;□i>=0;□i--,□j++)▼832.5
□□□□x[]〈↑↑↑↓↓↑←←→,j↓↓←j]=s[i]-'0';▼864.0
【註】@：思考中，〈：ＢＳ，◇：ＴＡＢ，□：空白，▼：リターン＋経過時間
```

図１　操作過程のプリントアウトの一部（学生Ｇ２，課題２）

3　結果と考察

3．1　被験者のグルーピングとプログラム作成過程

普段の演習風景から被験者となる学生が大まかに以下のグループに分けられることが分かっているが、データの分析方法によってこれらのタイプに差が現われるか調べた。なお、Ｇ１～Ｇ８は学生を特定するための仮の名前である。被験者10人中、2人については操作ミスによってデータの収集が行なえなかったため、省略してある。

タイプＡ　…　じっくり問題を分析してからコンピュータに向かう。プログラムの完成は早い。（Ｇ１）

タイプＢ　…　少しずつプログラムを入力しながら、途中でよく考える。（Ｇ２，Ｇ３）

タイプＣ　…　プログラムを段階的に考え、作成し、試行錯誤で完成させる。行き詰まったときに、じっくり考えずにプログラムを変更しては試すことを繰り返す。（Ｇ４，Ｇ５）

タイプＤ　…　Ｃ言語の文法とアルゴリズムが十分に身に付いていないため、作成途中で教官の助けを求める。（Ｇ６，Ｇ７，Ｇ８）

操作過程データのプリントアウトと操作を再現することによって、プログラミング過程の分析を行ない、各学生のタイプＡからＤによって、以下に述べるように、それぞれ差が現われることが分かった。

ここでは、課題２について各タイプの簡単な比較を行なってみる。アルゴリズムを考えなくてはならない難しい課題の方が差がでる傾向にあるようである。タイプＢとＣの学生は、プログラムの骨組みの作成、制限された条件での処理の作成・実行、動作の確認、一般条件での処理の追加、の順にプログラムを完成させていくが、途中で問題が生じたときの対処の仕方に違いが見られる。なお、※は全体的なコメントである。

タイプＡ（Ｇ１）…

(1) しばらく課題の紙を見て考えていた。
(2) 使用する関数の頭部のみ先に入力。
(3) 関数 set を課題のプリントから引用。
(4) 関数 add の入力。'for (' と入力したところで、上の行に戻って変数 i の宣言を行なっている。i は、両方の配列の要素数を越えないまで増やし続ける。
(5) ２つの配列の要素ごとの加算結果を別の配列 c に入れ、10 で除算した商と余りを求めている。商をキャリー（繰り上がり）としているが、加算の式にキャリーが含まれていないことに気付き、カーソルを戻して追加している。
(6) 対象となる数の桁数が異なるときの処理を追加。for の判定条件の追加など。
(7) 関数 main の作成。scanf で文字列として数値を読み込み、配列の数値に変換。
(8) 結果の表示部の作成。
(9) コンパイル、実行。何種類かの数値で動作の確認。問題なし。30 分で完了。
(10) 時間が余ったので、乗算も作成。乗算の作成だけで 50 分要した。

※プログラミングに取り掛かる前に方針・手法をよく考え、あたかも考えを整理するためにプログラムを作成しているようにさえ見える。他の課題についても全く同じである。

タイプＢ（Ｇ２）…

(1) 関数 main を値のセットと表示だけで作成。
(2) 関数 set を課題のプリントを参照して入力。
(3) コンパイル、実行。
(4) 関数 set のプロトタイプ宣言の追加。
(5) main に add の呼び出し、結果を表示する処理を追加。
(6) 関数 add の作成開始。for と c[i] = (a[i] + b[i])/10;
(7) 間違いに気付き、(a[i] + b[i])%10 に訂正。そして、キャリーの追加。
(8) 繰り上がりのない値で実行して動作の確認。値は関数 set の呼び出し時に引数で指定。
(9) 桁数が異なるときの処理を追加。短い方の数値の不足している桁の要素をゼロクリア。
(10) コンパイル、実行。何種類かの数値で動作の確認。問題なし。

※プログラムを土台から積み上げながら作っている。要所要所の動作を確実なものにしているので、途中で予期しない結果が出ても、原因を容易に見つけられている。

タイプＣ（Ｇ４）…

(1) main を入力し始め、途中で関数set と add のプロトタイプ宣言の入力。
(2) 中味のない関数 add の入力。関数 set を課題のプリントから引用。

(3) main の入力を継続。値を scanf で文字列に読むようにする。
(4) 結果表示部を入力しようとして、ヘッダファイル #include <stdio.h> の文を追加。
(5) コンパイルを行なったが、よく理解できないエラーメッセージのため、値をセットするために文字列を読み込んでいた scanf を削除。set の引数に直接文字列を記述。
(6) 再びコンパイルを行なったが同じエラー。関数 set のプロトタイプ宣言で引数が配列になっていないためであった。
(7) コンパイル、実行。そして、結果の最後に改行が表示されるように追加。
(8) 関数 add の入力。for と加算部を入力しようとして、変数の宣言を追加。
(9) 最初から桁数の異なるもの同士の演算を考えて作成。しかし、その方法と結果の桁数の求め方に苦労。
(10) 結果を入れる配列への代入でインデックスの順を逆にしてしまったので、最後に別の配列に逆順に代入。
(11) 加算のあとに % 10 を行なわなかったため、繰り上がりのある演算で結果の値が大きくなってしまっている。試行錯誤しても、このことに気付くことができなかった。未完成。

※プログラムの全体像が見えてしまったため、一息に作り上げようとしている。簡単な課題では完成させるのが一番早い方であるが、課題2のように考慮しなくてはならない事項が多いと、問題点の原因を明らかにするのに苦労して、対処療法的になってしまっている。

タイプD（G8）…
(1) 関数 set を課題のプリントから引用。
(2) main はプリントを参考に作成。add の引数と結果の表示の変数名の対応がとれていない。
(3) 関数 add の入力。加算される2つの変数への値のセット方法に苦労。getchar, scanf("%d",a[]) などと悩んでいる。結局、関数 add 中に関数 set の呼び出しを書いてしまった。set(a,123456);
(4) 配列の要素ごとの加算では、要素の終わりを判断するのに、NULL を使っている。文字列の処理で習ったことに影響された可能性も考えられる。
(5) キャリーを求めるのに、if(c[k]-'0'>=0) carry =1; とするなど、紙の上で筆算で行なうイメージを抽象化できていない。値の表現方法の理解が足りないようである。未完成。

表1　課題2のキー入力数

タイプ	A	B		C		D		
学　生	G1	G2	G3	G4	G5	G6	G7	G8
ファイルサイズ	669	848	915	937	702	747	565	590
アスキーコード	644	1005	943	1670	1482	828	888	898
%	31.2	18.1	28.7	26.8	19.6	30.9	26.4	33.5
カーソル（左右）	428	1427	705	1843	1859	702	829	1073
%	20.7	25.7	21.4	29.5	24.6	26.2	24.7	40.0
カーソル（上下）	809	2707	1368	2149	3230	772	1235	444
%	39.2	48.8	41.6	34.4	42.7	28.8	36.8	16.6
削　除	168	259	186	**436**	**772**	196	306	235
%	8.1	4.7	5.7	7.0	10.2	7.3	9.1	8.8
ヘルプ	1	0	0	10	9	0	17	0
%	0.0	0	0	0.2	0.1	0	0.5	0
メニュー（実行）	10	81	42	76	103	27	29	6
%	0.5	1.5	1.3	1.2	1.4	1.0	0.9	0.2
ＳＴＯＰキー	0	0	6	0	0	0	1	0
%	0	0	0.2	0	0	0	0.0	0
その他	3	67	41	56	109	153	55	26
%	0.1	1.2	1.2	0.9	1.4	5.7	1.6	1.0
全操作時間（分）	34	108	85	84	91	92	87	85

3．2　キー入力数と操作のカテゴリ分析

記録した操作過程データをプログラムによって加工、分析、図示することで操作過程の把握・評価が容易にならないか検討を行なった。

課題2について操作者のキー入力したキーの種類ごとにキー入力数と割合を求めると、**表1**のようになる。実際のプログラムになるアスキーコード入力数の割合に比べて、カーソル移動キーが非常に多いことが分かる。内容が難しい課題2の方がカーソルの上下移動の割合が多くなっているが、傾向としては課題1も同じである。また、タイプCの学生はアスキーコードと削除キーの入力が多く、プログラムを頻繁に書き替えていることが分かる。

次に、キー入力の種類から操作過程の状態を次のカテゴリーに分けて求め、分析を試みた。
(1)プログラム入力、(2)プログラム編集、(3)削除、(4)思考中、(5)関数の機能のヘルプ、(6)コンパイル・実行、(7)エラーのヘルプ。
「プログラム編集」は続けて5行以上のカーソル移動を行なった場合、「削除」は連続した5文字以上の削除または行の削除を行なった場合、「思考中」は30 秒以上キー入力の無い場合とした。

時間とともに操作過程の状態がどのように遷移していくかを調べた結果を**図2**と**図3**に示す。ある状態1から次の状態2まで線で結ばれた時間が状態1にかかった時間である。

タイプAの学生G1は、他のタイプの学生と比較して状態を遷移する頻度が明らかに少なく、連続して同じ操作を続けている。そして、最後にコンパイル・実行を行なって完成させている。タイプBの学生はプログラム入力と思考の間の遷移が多い。タイプCの学生もBと似ているが、課題2については編集への遷移が多い。

課題2について操作状態のカテゴリーごとの頻度と、最初にそのカテゴリーに遷移した時間（記録開始時からの時間）を**表2**に示す。タイプCの学生はキー入力の状態数が多く、延べ思考時間が少ない。よく考えずに対処療法的に対応している結果と考えられる。一方、タイプBとDの学生は延べ思考時間が長い。タイプBは途中で分からなくなった時に、紙の上に図を書くなどして頭の中を整理していたが、タイプCは教科書などに似たようなプログラムがないか探したり、文法を調べていることが多かった。課題1についても同様である。

図1に示す操作過程のデータと操作の再現によって、何につまずいたか等の細かい分析が可能であるが、これは紙の上にプログラムを書かせ、観察する方法と同様にC言語の "{" の対応、セミコロンの入力忘れなどの微視的なことに分析者の目が向いてしまう傾向がある。課題に対してどのように臨んでいるか、対処療法的になっていないかは**図2**，**図3**，**表2**のように操作状態をカテゴリーに分けて取り出すことによって、よりはっきりと把握することができる。分析には、このように全体を巨視的に把握する方法と、微視的に見る方法の両方が必要であるといえる。

3．3　プログラミングの環境について

コンパイル・実行の頻度が高いタイプBとCを見ると、コンパイルエラーとif文の条件が逆などの簡単な論理的ミスによることが多い。コンパイルエラーの文章が適切でないときは、原因を見つけるのに試行錯誤を繰り返している。このようにプログラミング作業を行なう環境によって影響を受けている面もあり、環境についても考慮する必要がある。

プログラミングに関係する環境として、エディタ、コンパイラ、C言語の3つの影響について収集した操作過程のデータから考えてみると、

(1) カーソル移動キーの入力が多いのは、エディタで一度に見られる行数が18行と少ないためで、マルチウィンドウのエディタで変数宣言部と実行部の記述を同時に表示して入力を行なうことができれば、カーソル移動が減ると考えられる。

(2) 文の終わりにセミコロン（；）を入れ忘れると、その先の文で理解できないエラーメッセージが表示され、原因が分からないために思い付きで各所に変更を加えてしまい、苦労している例が多い。コンパイラがもう少し適切なメッセージを表示してくれれば改善されると考えられるが、学生の経験不足で想像できていないともいえる。

(3) C言語については、データの抽象化が弱いため、配列同士の代入や比較が演算子を用いて行なえず、直感的に理解しにくく、学習者の理解の妨げになっている。また、文字列の型がないのに、変数宣言では一見文字列定数の代入ができるようになっているため、混乱を招いている。

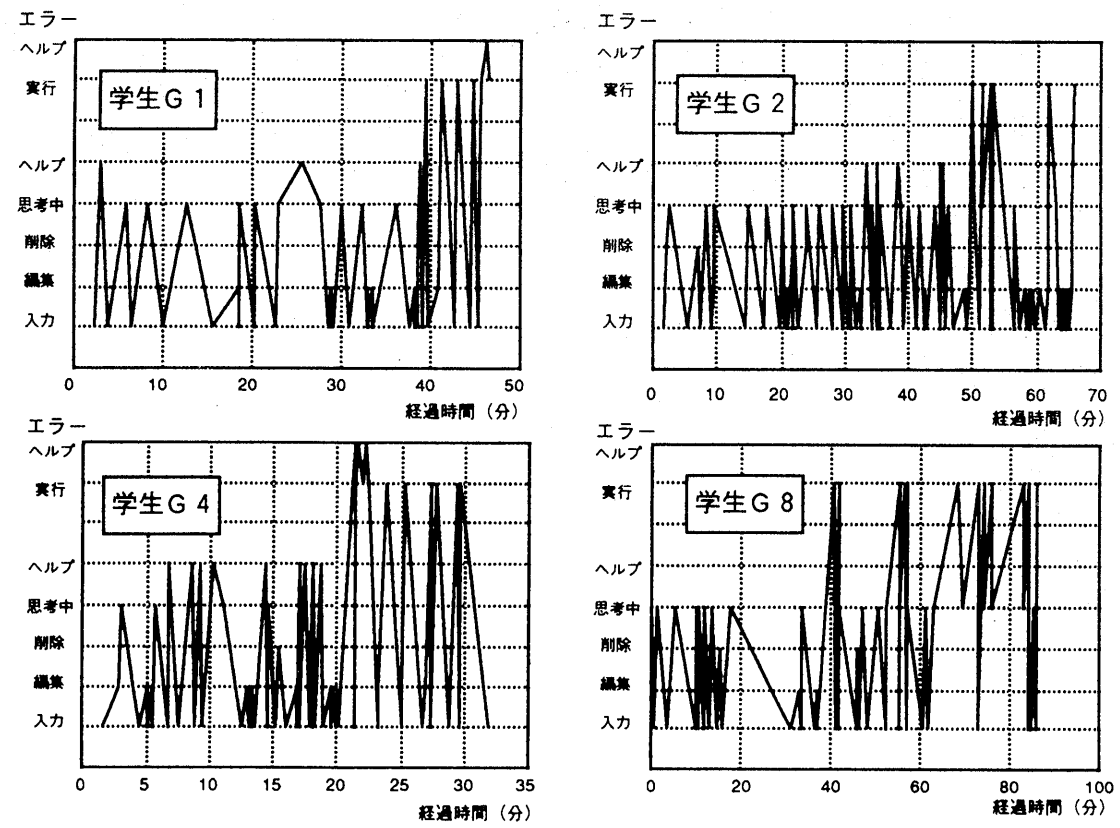

図２　課題１の操作状態の遷移

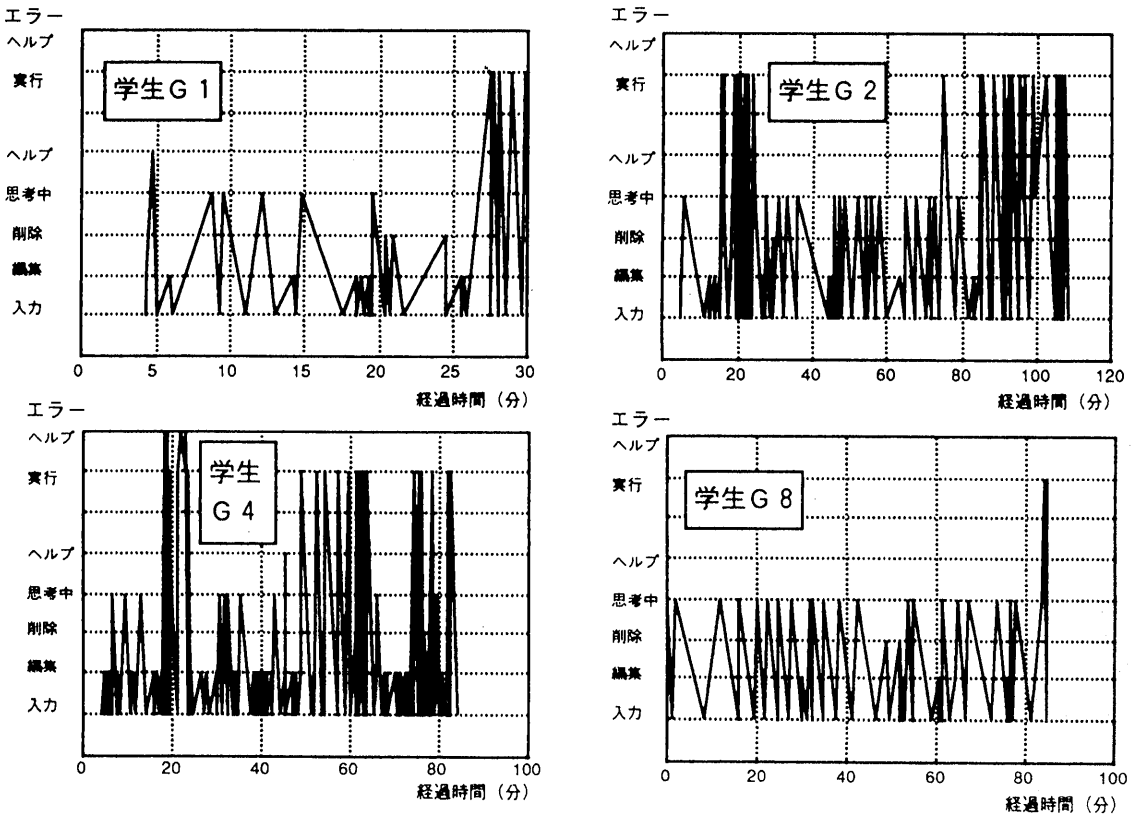

図３　課題２の操作状態の遷移

表2　課題2の各操作状態の頻度

タイプ	A	B		C		D		
学　生	G1	G2	G3	G4	G5	G6	G7	G8
プログラム入力	22	59	41	91	96	40	46	31
開始時間（分）	4.3	4.9	7.3	4.1	4.2	3.3	4.2	0.8
編集	9	26	16	51	45	11	19	7
開始時間（分）	4.3	5.0	22.4	4.5	7.7	9.0	9.8	30.0
削除	3	2	5	12	8	3	3	3
開始時間（分）	20.4	29.3	28.5	25.4	18.0	3.8	48.6	48.9
思考	5	25	17	10	17	25	19	21
開始時間（分）	8.8	5.1	7.8	6.6	12.7	10.0	7.8	1.8
最大思考時間（分）	2.6	5.8	7.3	2.5	1.6	6.0	5.7	6.5
延べ思考時間（分）	6.5	35.5	28.0	**9.7**	**15.7**	34.9	29.8	40.4
関数のヘルプ	1	0	0	2	1	0	1	0
開始時間（分）	4.6	-	-	19.3	9.9	-	35.3	-
コンパイル実行	5	24	15	26	42	11	12	3
開始時間（分）	27.5	15.1	56.4	18.0	9.4	67.6	41.2	84.2
エラーのヘルプ	0	0	0	4	8	0	0	0
開始時間（分）	-	-	-	18.1	9.6	-	-	-

4　おわりに

コンピュータの操作を記録したデータについて、時間軸で個々の操作を追跡する微視的な分析と、操作の状態をカテゴリに分けて学習者の大きな行動を見る巨視的な分析を行なうことで、プログラミング過程を詳細に把握することができた。

この方法によって、学習者がどのようなことにつまずき、何が苦手なのか、また、簡単な制約条件のものから試作していくなど、どのようにプログラムを組み立てているのかが分かる。

全操作を追跡する微視的な分析では、時間がかかる上に、大きな流れを見失いがちであるが、カテゴリ分析を使った巨視的な方法を先に適用することで、状態の遷移具合等から学生が解決への道順を見失ってしまった時の行動等をみつけられ、後の微視的分析を短時間に効率的に行なうことができる。

そして、学生と道順を間違えたときの考え方などを話し合う等、学生にフィードバックをしていくことで、きめの細かい指導が行なえると考える。

今回の分析の具体例では、文法をマスターしたプログラミングが得意なタイプCのような学生でも、アルゴリズムを考える必要があるときに十分に考えないと、つまらないことで混乱に陥ってしまうことがはっきりと現われた。このような学生には、始める前とプログラミング中に行き詰まった時に、紙と鉛筆で図を書いて考えさせることで、理解を深めさせることが可能になると考える。

今後、構造化データなどを使う課題で、分析結果を学生にフィードバックしながら、きめの細かい指導を実現する手法について検討していく予定である。

本研究は文部省科学研究費補助金一般研究C（No.05680172）「プログラム作成過程に視点を置いた評価法の開発」の研究成果の一部である。

参考文献

[1] カーニハン，リッチー著，石田晴久訳：プログラミング言語C 第2版，共立出版，1989

[2] 前田恵三，中野靖夫：コンピュータ操作過程の再現，日本教育工学会雑誌，Vol.16, No.4, 1993

研究発表－３

コンピュータ・プログラムの作成に関する
教育的評価次元の構成

田　中　　敏 *

（平成６年10月31日受理）

要　旨

コンピュータ・プログラムの作成が学習・問題解決スタイルをプランナーからマニピュレータのスタイルへ移行させていると A. Minskey は示唆している。後者のマニピュレータ・スタイルを評価するため、本研究はコンピュータ・プログラムの作成作業を分析することによっていくつかの評価次元を構成しようとした。166 人の大学生にプログラミングまたはデバッギングの二者択一的やり方（プログラムの実行結果は変わらない）を提示し、両者の一方に対する好みを５ポイント尺度上に評定するよう求めた。この評定値について因子分析をおこない、次の４つの次元を抽出した。１．新奇事項への着手の回避（avoidance to do anything new）、２．プログラムの表記と内容の明確化（keeping clear view of descripion and contents of program）、３．その場しのぎ・不拡大方針（makeshift and inextension policy）、４．探究・発展の志向（intention to progress）。引き続き分析した結果、プログラム読み書き能力の高い者ほど次元２と次元３に大きなマイナスの負荷量を示した。以上の結果はマニピュレータの学習スタイルという観点から考察された。マニピュレータとは活動の「正しさ」よりも「うまくゆくこと」に関心をもつ者である。

KEY　WORDS
computer programming　コンピュータ・プログラムの作成
planner and manipulator　プランナーとマニピュレータ
educational evaluation　教育評価　　factor analysis　因子分析

問　題

コンピュータの利用者は２層に分かれる。ひとつはアプリケーション・ユーザーの層であり、もうひとつはプログラマーの層である。前者は既製のプログラムを使用する。後者はそのプログラムを作成する。

情報教育と情報産業の動向は、アプリケーション・ユーザーに特別のリテラシーを求めることを極力避けようとしている。しかし、あいかわらずプログラマーには、人間の知性ではなくハードウェアのアーキテクチャーに対して特殊化された高コストのリテラシーを要求する。

ここに、それまでの言語的リテラシーを用いる学習者とはまったく異なる、新しいタイプの学習者が見いだされることになった。その典型はミンスキー（Minsky, A., in ブランド, 1988）

* 教育方法講座

が従来のブルーナー（Bruner, J. S.）流の学習者像である「プランナー」(planner) に対比させた「マニピュレータ」(manipulator) である。学習場面において、プランナーは仮説を立て、それを検証し、修正し、だんだんと「正しいこと」に接近してゆく。これに対して、マニピュレータは正解を発見しようとするよりも誤りを発見しようとする。そして、その誤りを取り除くことによって「うまくゆくこと」に接近してゆく。さらに対比的にいえば、プランナーは自分の仮説にしたがって計画的に環境にはたらきかけ、現実を正しく認識し、支配しようとするが、マニピュレータはその時々の事情に応じて臨機に予定を変更し、現実との妥協をはかるにもやぶさかではない。要するに、マニピュレータは問題の最終的・決定的な解決を意図していない。それに代えて、その場面に問題が起こらないようにし、起こったら即時対応するというようなプラグマティックな処理方針をとる。

「このマニピュレータの発想はコンピュータ・プログラムの作成における誤り除去の作業に由来している。コンピュータ・プログラムのどこかに誤りがあることがわかると慣用的に『プログラムのなかにバグ（虫）がいる』と表現するが、このバグに『除去』の意味をもつ接頭辞 “de-” を付加してプログラム中の誤りを除去することを『デバッギング』(debugging : 虫取り）と呼んでいる。デバッギングはプログラムの作成に宿命的に付いて回る面倒な作業であり、プログラミングとはデバッギングの別の名前である。‥（中略）‥プログラムの作成に誤りがつきものであり、デバッギングが不可避であれば、最初から正しいプログラムを作ろうなどと考えないほうがよい。正しいかどうかということよりも、その時々の状況に応じて少しずつ良くしてゆこうと考えるほうが適応的である。このデバッギングの作業方針がマニピュレータとしての学習者像の基礎にある。マニピュレータの原意は『手作業する人』であるが、デバッギングは不良箇所を見つけてはひとつひとつ繕ってゆくという、まさに手作業である。』（田中, 1994, p. 94）

こうしたマニピュレータの実務的な作業方針は、プランナーの計画的な作業方針に比べて首尾一貫性に欠けるが、その分、不測の事態によく対処することができる。そして、現実に計画外の不測の事態というものは必ず生じるものである。それなら、計画とは必ず失敗するものなのだと最初から決めてかかるほうがよい。究極の計画、万能の知識、最終の解決は存在しないか、存在しても、今すぐわれわれの手の届くところにあるものではない、と。

マニピュレータはひとつの新しい学習者像を示すにとどまらず、もっと一般的な活動スタイルの変化を表している。この変化は時代的・社会的に見れば、「システマティク処理からプラグマティク処理へ」と定式化することができる（田中, 1994, pp. 116-120 参照）。この後者への大局的変遷の結果をとらえるのに、従来の達成型・問題解決型の学習者像に関する観点では不十分であることが容易に予想されるであろう。そこで、本研究は、マニピュレータとしての学習者における活動特性をとらえるための基礎的アプローチを試行しようとする。

このため、マニピュレータの考え方が端を発するコンピュータ・プログラムの作成作業を取り上げ、その潜在次元を抽出してみる。対象は複数のコンピュータ実習の授業を修了したか、受講中の大学１年生であり、実習内容がプログラミングに統一されていないためコンピュータの利用経験はあってもプログラムの作成に関してはまったくの初心者から一応の理解者まで連続的分布が期待できるサンプルとした。したがってまた制約があるとすれば、本研究が明らかにしようとするプログラムの作成の諸次元は初学者に限って存在するものであり、その意味においてコンピュータ教育上の初期の評価次元として適用されるものである。

調　査

目　的

コンピュータ・プログラムの作成作業を支配する次元を抽出する。大学１年生の初学者を対象に、質問紙において作業場面を例示して当該作業への同調傾向を調べ、因子分析法を用いて妥当な次元を探索する。

方　法

対象者　コンピュータ実習の授業（半期）を修了したか受講中の大学１年生 175人。実習内容は各授業によってプログラミングをはじめとして、ワープロ、グラフィック、表計算ソフトの利用、キーボードのタイプ練習など多岐にわたるので、履歴ではなくプログラムの理解度によって既存適性を把握することにした（以下の『質問紙』の項参照）。

質問紙　ＢＡＳＩＣのプログラムの作成を課題として仮定した（表１参照）。そして、このプログラムの作成時に生じる作業場面を15個想定した（表２参照）。これらの作業場面は、今回の対象者とは異なる初学者が実際にプログラムの作成を実習したときの「キータッチ・キャリア」[1]の記録を再生検討して構成されたものである。教示文は次のとおりであった。「あなた自身が上のようなプログラム（表１の解答例）を作るとします。そのときに、あなたならば『こうするだろうな』ということを想像して、以下の質問に回答してください。なお、質問はひとつひとつ調査者が読み上げて説明します。その説明が終わってから回答するようにしてください。また、『正しい回答』『間違った回答』はありませんので、思ったとおりを答えるようにしてください。」

表１　プログラム作成の問題と解答例

［問　題］
Ｘ＋Ｙ の足し算をするプログラムを作り、
３＋５ の答えを画面に表示せよ。

［解答例］

```
10  INPUT  X
20  INPUT  Y
30  PRINT  X+Y
```

この教示の後、表２のそれぞれの作業場面について「あなただったら、そうしますか」とたずねた。対象者の回答は、「ぜったいそうしないと思う」から「必ずそうすると思う」まで５段階尺度において求めた。なお、以上の作業場面のほかに、表１解答例のプログラムが理解できるかどうかを「ぜんぜんわからない」「少しだけわかる」「だいたいわかる」「完全にわかる」の４段階においてたずね、また、コンピュータを使う作業が好きかどうかを「ぜんぜん好きでない」から「たいへん好きである」まで５段階においてたずねた。

手続き　調査は集団調査であり、300 人規模の大教室において実施された。対象者に質問紙を配布し、そこに書かれた課題を実験者が教室前面の２台のＴＶ画面において実演して見せた。実演は表１解答例のプログラムを作成し、実行して、数値を入力し、演算結果を得るというものであった。その後、質問紙の教示文を読み上げ、実験者がひとつずつ作業場面を説明しながら、そのたびごとに対象者の回答をうながし、質問紙の評定尺度上に書き込ませた。

表2　質問紙の項目（プログラムの作成作業場面）

［項目１］
プログラムを書くとき、１行ずつ書いては、すぐに実行してみた。

［項目２］
```
10 INPUT X
20 INPUT Y
30 PRINT X+Y
```
の10行と20行を一緒にし
```
10 INPUT X: INPUT Y
20 PRINT X+Y
```
と書いてもよいと言われたので、そう書くことにした。

［項目３］
```
10 INPUT X
20 INPUT Y
30 PRINT X+Y
```
いったん　X＋Y　の答えを単独で求めておきたい。
その行を　ＫＯＴＡＥ＝Ｘ＋Ｙ　として途中に加えることにした。
```
10 INPUT X
20 INPUT Y
30 KOTAE=X+Y
40 PRINT KOTAE
```

［項目４］
```
10 INPUT X
20 INPUT Y
30 PRINT X+Y
```
の途中に KOTAE=X+Yを加えるとき、
新たに25行として加えることもできた（右）。
```
10 INPUT X
20 INPUT Y
25 KOTAE=X+Y
30 PRINT KOTAE
```
しかし、すべての行番号を10飛びに付け替えることにした（下）。
```
10 INPUT X
20 INPUT Y
30 KOTAE=X+Y
40 PRINT KOTAE
```

［項目５］
ＫＯＴＡＥ＝Ｘ＋Ｙ　と書くところは、
単に　Ａ＝Ｘ＋Ｙ　でもいいはず（左）。
　けれども、やはり
ＫＯＴＡＥ＝Ｘ＋Ｙ　と書いた（右の枠）。
```
10 INPUT X
20 INPUT Y
30 A=X+Y
40 PRINT A
```
```
10 INPUT X
20 INPUT Y
30 KOTAE=X+Y
40 PRINT KOTAE
```

［項目６］
一応、プログラムの全体を書いた。すぐに実行してみた。途中でエラーが出ても、そのとき直せばいいや、と考えて、実行前に１行ずつ見直してみたりすることはしなかった。

［項目７］
ＰＲＩＮＴ　と書くところを　ＰＲＮＩＴ　と打ち間違えていた。
ＮＩ　だけを直せばいいのだが、頭の　Ｐ　から全部打ち直した。

［項目８］
INPUT と書くところを INPT としか打ってなかった。T の直前に　I を挿入すればいいし、挿入のキーがあることもわかっていたが、結局、最初の　I からその行全部を打ち直した。

（表２の続き）

［項目９］

先生から、「実行する前には必ずセーブせよ」（プログラムをディスクに保存せよ）と言われていた。思い出して、実行前にセーブした。

［項目10］

プログラムを実行した。　３　と　５　を入力したら　８　と出た。
これで完成と思ったが、ほかにも、　－３　と　－５　とか、
０．３　と　０．５　とか、いろいろと試してみた。

［項目11］

プログラムを実行したら、

```
? ■
```

と聞いてきた。
わざとエラーを出させてみたくなり、数字ではなく、なにか文字を入力することにした。
そこで、でたらめに　ＡＢＣ　と入力してみた。

［項目12］

```
10 INPUT X
20 INPUT Y
30 KOTAE=X+Y
40 PRINT KOTAE
```

の30行と40行を一緒にして

```
10 INPUT X
20 INPUT Y
30 PRINT KOTAE=X+Y
```

とすれば、
ＫＯＴＡＥ　も計算できて、プリントもできるのではないかと思い、やってみた。

［項目13］

```
10 INPUT X
20 INPUT  Y
30 KOTAE=X+Y
40 PRINT KOTAE
```

の10行のＸと、20行のＹの位置がずれていた。
実行上はまったく支障がなかったが、そろえることにした。

［項目14］

30行めは、純粋な計算ステップであるから、ほかの行と区別して、段を付けてみた。右のように２字分、下げた。

```
10 INPUT X
20 INPUT Y
30    KOTAE=X+Y
40 PRINT KOTAE
```

［項目15］[1)]

このプログラムは　３　と　５　をひとつずつ入力していたが、
３＋５　といっぺんに入力できるプログラムが出来ればいいのにな、と思った。
　そう先生に言ったら、「足し算だけなら、これで充分じゃないか」と言われた。
しかし自分自身は、充分だという気はしなかった。

[1)] 項目15は「あなただったら、充分だという気がしますか。」というたずね方をした。

結　果

回答不備の９人のデータを削除したので、最終的に分析対象は 166人になった。

１．作業場面の項目に対する評定得点についての因子分析

作業場面の項目に対する評定は、それぞれの例示された作業について（自分ならば）「ぜったいそうしないと思う」を１点、「たぶんそうしないと思う」を２点、「どちらともいえない」を３点、「たぶんそうすると思う」を４点、「必ずそうすると思う」を５点として得点化した。この評定得点の平均と標準偏差を表３に示す。

表３　各項目の評定得点の平均と標準偏差（N=166）

項目	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M	2.23	2.96	2.70	3.36	2.49	3.23	2.49	2.39	4.22	3.44	2.55	2.57	3.65	2.17	2.91
SD	0.86	1.01	0.96	1.07	0.99	1.05	1.14	1.13	0.81	1.01	1.01	0.92	1.03	0.75	0.94

ここで、項目９「プログラムを実行前にセーブする」は平均＋標準偏差の値（5.03）が上限値の５点を越えていたので天井効果が生じたとみなし、以下の因子分析の対象からは除外することにした。

主成分分析の結果、４因子を適当とみなして抽出した。バリマクス回転後、表４の因子パターンを得た。

±.35 以上の因子負荷量を示した項目の内容を参考に各因子の解釈を試みた。

因子Ｉは項目７・項目８（誤って綴られた語を初頭からすべて打ち直す）より、挿入キーや削除キーなどによる他の機能を使わずに少数の単一のキー操作だけで済ませようとする傾向ではないかと考えられる。それを裏づけるように因子Ｉは項目２（先生にいわれた工夫をやってみる）と項目12（文法に違反するかもしれない書き方を試みる）にはマイナスの負荷をあたえている。つまり「新奇な、不慣れなことには手を出さない」「いま覚えようとしている範囲内で手一杯である」というような、習いたての初心者に特有の余裕のない状態であると思われる。したがって因子Ｉは『新奇事項への着手の回

表４　バリマクス回転後の因子パターン[1)]

項目番号	因子Ｉ	因子II	因子III	因子IV	共通性
7[2)]	0.832[3)]	0.130	0.114	0.063	0.726
8	0.819	0.078	0.022	0.002	0.677
14	-0.049	0.665	0.104	0.040	0.457
5	-0.001	0.585	0.132	0.190	0.396
3	0.230	0.511	0.050	0.174	0.347
12	-0.397	0.446	0.001	0.345	0.476
4	0.145	0.327	0.131	-0.163	0.172
6	0.108	-0.046	0.733	0.051	0.554
1	0.007	0.156	0.679	-0.054	0.488
2	-0.387	0.153	0.423	0.306	0.446
10	-0.157	0.395	-0.426	0.394	0.517
11	-0.004	0.100	-0.366	0.684	0.612
13	0.027	0.283	-0.241	-0.422	0.318
15	-0.042	0.112	-0.094	-0.578	0.357
固有値	1.785	1.683	1.621	1.452	

1) 全分散に対する４因子の累積説明率は 46.7% 。

2) 項目内容については表２参照。

3) 下線を付した負荷量は .350 以上。

避』（avoidance to do anything new）と命名した。

次に、因子Ⅱは項目14（計算ステップの行を段を付けて書く）・項目５（変数名をＡとするよりもＫＯＴＡＥとする）・項目３（計算と表示を分けて独立の行とする）・項目12（文法に違反するかもしれない書き方を試みる）・項目10（課題の３と５以外にも－３と－５などを入力しプログラムの作動を点検する）より、『（プログラムの）表記と内容の明確化』(keeping clear view of descripion and contents of program）と命名した。

続いて、因子Ⅲは項目６（プログラムを書き上げたらよく見直さずに即実行する）・項目１（１行ずつ書いては実行してみる）・項目２（先生にいわれた工夫をやってみる）より、場当たり的で、その場しのぎの操作をおこなっているように推測される。そのような場合は、課せられたもの以上に作業を拡大したり応用したりしないものである。それゆえに、おそらく因子Ⅲはマイナスの負荷を、項目10（課題の３と５以外にも－３と－５などを入力してみたくなる）と項目11（プログラムの実行中にわざとエラーを出させてみたくなる）にあたえるのであろう。したがって因子Ⅲは『その場しのぎ・不拡大方針』（makeshift and inextension policy）と命名した。

最後の因子Ⅳは、上述の項目10と項目11にプラスの負荷をあたえ、項目13（同じ内容の行について形式的なケタ揃えをする）と項目15（プログラムの機能を課題の要求以上に高める必要はない）にマイナスの負荷をあたえることから、現状にとどまらずに探究・発展を進めようとする傾向が読み取れる。そこで因子Ⅳは『発展・探究の志向』(intention to progress）と命名した。なお、この因子Ⅳは項目12（文法に違反するかもしれない書き方を試みる）にも、ある程度のプラスの負荷（.345）をあたえることからも、その内容が例証されるであろう。また、特に項目10と項目11に対しては、上掲の因子Ⅲ『その場しのぎ・不拡大方針』と逆方向の負荷をあたえているので、因子Ⅲと部分的に相反するものとして因子Ⅳを『発展・探究の志向』と解釈することは適当であろう。

総じて、多少の重構造を示したが因子の解釈は明快であり、妥当性は高いと思われる。

項目の側からいえば、項目４が他の項目と比べて著しく共通性が低く（.172）、独自に変動する項目であると考えられる。そこで項目４は後に単独で取り上げて分析することにした（後出３参照）。

２．プログラムの理解レベルとコンピュータ作業の好き嫌いによる因子得点の比較

本研究の目的からして、以上の４因子がコンピュータ・プログラムの作成に及ぼす影響は対象者の適性に応じて変化するものであることが期待される。このことを確かめるため、各因子得点について、適性の異なる対象者を比較してみることにする。

質問紙において対象者に、表１解答例のプログラムをどれくらい理解できるかを４段階でたずねたが、この回答をそのままコンピュータ・プログラムに関する理解レベルとし、低いほうから『レベル１群』『レベル２群』『レベル３群』『レベル４群』に対象者を分けた。同様に、コンピュータ作業に対する好き・嫌いの程度を５段階でたずねたが、その回答において「ぜんぜん好きでない」と「あまり好きでない」を『嫌い群』、「どちらともいえない」を『中立群』、「好きである」と「たいへん好きである」を『好き群』とした。なお、全体の平均は理解レベルが 2.49（0.98）、 好き嫌い程度が 3.57（1.65） であった（カッコ内は標準偏差）。なお、両者のクロス集計の人数は表５の Ｎ に示されるが、特に有意な偏りは見いだされなかっ

た（$\chi^2_{(6)}=5.35$, n.s.）。このことから、コンピュータ・プログラムの理解とコンピュータ作業の好き嫌いは直接には関連しないといえる。

表5は理解レベル群と好き嫌い群における標準因子得点の平均と標準偏差を示したものである。それぞれの因子について、4（理解レベル）×3（好き・中立・嫌い）の分散分析をおこなった。結果として、いずれの因子についても交互作用はまったく検出されなかった。なお、以下の分散分析において主効果が有意であった場合、平均間の多重比較にはテューキ法を用いた。

表5　プログラムの理解レベルとコンピュータ作業の好き嫌いによる標準因子得点の平均と標準偏差

レベル		1			2			3			4		
嫌・好		嫌	中	好	嫌	中	好	嫌	中	好	嫌	中	好
	N	7	8	13	17	17	25	9	11	29	6	5	19
因子Ⅰ	M	0.39	0.20	-0.17	0.49	0.21	-0.20	0.41	-0.05	-0.32	0.23	0.04	-0.25
	SD	0.96	0.91	0.84	1.04	1.11	0.97	1.27	1.07	0.81	1.21	1.21	0.96
因子Ⅱ	M	0.11	0.39	0.37	-0.10	-0.34	0.11	-0.21	0.30	0.21	-0.94	-0.37	-0.22
	SD	0.64	1.29	1.14	0.73	0.84	1.18	0.61	0.77	0.81	0.98	0.73	1.35
因子Ⅲ	M	0.45	0.59	0.29	0.50	0.26	0.04	-0.23	0.14	-0.23	-0.18	-0.70	-0.73
	SD	1.20	0.96	0.85	0.78	0.99	1.09	0.89	0.78	1.14	0.60	0.87	0.64
因子Ⅳ	M	0.45	0.29	0.37	-0.52	-0.18	0.09	-0.12	-0.36	-0.04	-0.33	0.05	0.40
	SD	0.69	1.04	0.74	1.02	1.01	0.88	0.49	0.81	1.11	0.74	0.97	1.34

因子Ⅰ『新奇事項への着手の回避』については、好き・嫌いの主効果が有意であった（$F_{(2,154)}=4.78$, $p<.01$）。多重比較によれば、嫌い群の平均（0.42）が好き群の平均(-0.24）よりも有意に大きかった（MSe=0.98, $p<.05$）。中立群の平均（0.12）は両者の中間にあり、全体として直線的に変化していた。このことから、コンピュータ作業が嫌いであるればあるほど、新奇の操作がたとえ合理的・節約的であると知っていても、その利用と学習を回避する傾向が強くなるといえる。

因子Ⅱ『表記と内容の明確化』については、理解レベルの主効果が有意であった（$F_{(3,154)}=2.96$, $p<.05$）。各群の平均は大きい順に0.31（レベル1群）、0.16（レベル3群）、-0.08（レベル2群）、-0.39（レベル4群）であり、多重比較によればレベル1群とレベル4群の間に有意差があった（MSe=0.97, $p<.05$）。したがって初級者は因子Ⅱに強く支配される傾向があり、各行の内容をわかりやすく書き表し、よりよく把握しようとするが、これに対して、上級者は今回のプログラムのように全体の流れが明白な場合、あるいは実行上の誤作動が生じない場合には、プログラムの表記と内容に特別の注意をはらう必要性を感じないのではないかと考えられる。

因子Ⅲ『その場しのぎ・不拡大方針』についても理解レベルの主効果が有意であった（$F_{(3,154)}=5.52$, $p<.01$）。各群の平均は大きい順に、0.41（レベル1群）、0.24（レベル2群）、

-0.15（レベル3群）、 -0.62（レベル4群）と直線的に並んだ。多重比較の結果、レベル1群とレベル4群との間に有意差があった（MSe=0.90, p<.05）。 このことから、理解レベルの低い者の作業スタイルはいっそう場当たり的であり、課せられた問題だけに限定して作業を済ませてしまうおうとする傾向が強いといえる。これと対照的に、理解レベルの高い者は、プログラムの実行に際していろいろなケースを試みたり、わざとエラーを求めてみたり、プログラムの適用と作動を試行・点検することに動機づけられるように思われる。

最後に、因子Ⅳ『探究・発展の志向』については、理解レベルの主効果が有意傾向であった（$F_{(3, 154)}=2.29$, p=0.081）。しかしながら、多重比較の結果はいずれの群間にも有意差は見いだされず、実質的な差は生じていないと判断すべきであろう。ちなみに、平均の大きいほうから並べると、レベル1群（0.36）、レベル4群（0.20）、レベル3群(0.13）、レベル2群（-0.17）であり、群の順位にも取り立てて一貫した傾向を読み取ることはできない。

3．項目4についての分析

項目4の内容は「新しい行を途中に設けるとき改めて行番号を10飛びに付け替える」という作業である。評定得点を従属変数とした4（理解レベル）×3（好き・中立・嫌い）の分散分析の結果、理解レベルの主効果のみが有意であった（$F_{(3, 154)}$ 4.31, p<.01）。 テューキの多重比較によれば、レベル1群（3.51）・レベル2群（3.53）・レベル3群（3.45）と、レベル4群（2.77）との間に有意差があった（カッコ内は群平均, MSe 1.10, p<.05）。

基本的に、ＢＡＳＩＣのプログラミングでは最初の行番号はプログラムの作成が終了するまで付け替えないのが常識であり、まさにそのために最初から10ずつ飛び飛びに付け、新しい行は1から9までの中間番号を付けるようにしている。また、たとえ、このことを知っても知らなくても、今回のプログラムに新たな1行を挿入する際の全行番号の付け替えはまったく不要であり、たんに冗長で面倒なだけである。したがって、理解レベルの低い者はプログラムの外見や表記の形式にとらわれる作業傾向が強いということが示唆される。このことの傍証として、項目4は低いながらも因子Ⅱ『表記と内容の明確化』に対して相応のプラスの負荷量（.327）を示していることが注目される。

考　察

以上の調査結果に基づいて、プログラム作成者の具体的な像を描き出すように考察を進めてみたい。そうすることは、プログラム作成者の諸作業に対する教育的な評価次元を構成することにもなるであろう。

まず、因子Ⅰ『新奇事項への着手の回避』について、コンピュータ作業の嫌いな者がその傾向を強くもつことがわかった。一般的にいえば、この因子は学習領域や学習材料に対するネガティヴな態度であり、学習意欲・練習意欲・試行意欲の見られない活動を引き起こすことになる。その際、特に注意すべきは、因子Ⅰについて好き・嫌い群の主効果のみが見られ、理解レベルは有意な変動を示さなかったということである。したがってコンピュータ作業が嫌いとなると、コンピュータ・プログラムに関する理解レベルが高かろうと低かろうと関係なく新奇事項に対する敬遠が起こりうる。教育的には、理解レベルの高い群が問題であり、プログラムの

理解力や作成力が優れていても新奇事項の取り込みにおっくうな者が存在する。プログラムの作成作業に対する評価次元としては因子の名称を内容的に逆転させて『新奇事項への着手』とし、その程度を評価すればよい。

次に、因子Ⅱ『（プログラムの）表記と内容の明確化』と因子Ⅲ『その場しのぎ・不拡大方針』については、両者とも理解レベルの主効果が有意であり、共にレベル1群とレベル4群との間に差異が見られたので一緒に検討すべきであろう。すなわち、理解レベルが上昇するにつれて、『表記と内容の明確化』と『その場しのぎ・不拡大方針』が並行して低下するのである。このことは統合的にみれば、プログラムの表記と内容と課題達成を重視する作業者像から、プログラムの機能と拡大適用を重視する作業者像への移行を示唆していて興味深い。この移行は、冒頭の『問題』において述べた、プランナーからマニピュレータへの学習者像の変遷と同質のものではないだろうか。

ある意味で、プログラムの表記と内容を明確に書き表すことは処理の「正しさ」を認識しようとすることである。それはプランナーが正しい仮説を求めようとすることと同じである。しかし、そのような作業にこだわらなくなること（因子Ⅱの負荷の低下）はプランナーからマニピュレータへの移行を示しているといえる。マニピュレータは、プログラムの中身の「正しさ」を認識することよりも、プログラムの機能（うまくゆくかどうか）を点検するほうにずっと関心がある。このためまたマニピュレータは、因子Ⅲによる項目11（プログラムの実行中にわざとエラーを出させてみたくなる）の負荷に典型的に表されているように故意に失敗を求めてみたり、当面の課題達成にとどまらずプログラムの適用を広げてみたりする。

かくして、理解レベルの高い者に見られる因子負荷特性はマニピュレータの像を描き出すように思われる。

この点、最後の因子Ⅳについて理解レベルの効果が見いだせなかったことも示唆的である。通常の学習の進歩は因子Ⅳ『探究・発展の志向』と正の相関を示し、問題が解決されても、さらに解決の改善や応用へと向かうであろうが、コンピュータ・プログラムの作成は当面の問題が解消されれば特にそれ以上の問題を（欲してもよいが）欲しなくてもよい。プログラムに多少の改善の余地があっても、重大な問題を起こさなければ十分であるといえる。なぜなら、マニピュレータの活動方針と同様にコンピュータ・プログラムの作成も、（起こりうるすべてのバグの発生を予知することが不可能であるゆえに）問題がとことん解決された状態を最良とすることがむずかしく、たとえ問題が残っていても現実に問題が起こっていない状態をもって「よし」とせざるをえないような性質の作業にほかならないからである。

以上、因子Ⅱと因子Ⅲは、マニピュレータという観点に立った特有の評価次元を構成することができるが、最後の因子Ⅳはコンピュータ・プログラムの作成に限定されない資質を評価するものと考えるべきであろう。

そこで、因子Ⅱと因子Ⅲの内容を、コンピュータ・プログラムの作成力・理解力とポジティヴな関係をもつように書き換えて教育的な評価次元を構成すれば、『プログラムの表記と内容に対する機能の優先』と『適用の試行と点検』となる。因子Ⅳはこのまま『探究・発展の志向』でよいと思われるが、おそらく、その評価次元はコンピュータ・プログラムの作成作業を弁別的に価値づけることができないであろう。

なお、今後の課題として次の点を考慮した追試が必要である。①質問紙において提示する作業場面の種類と個数を増やすこと。②対象者にもっと高度の能力をもつプログラマーを含める

こと。③対象者の通常領域の問題解決力と活動方針を調べて、それがコンピュータ・プログラムの作成力と作業方針に関連するかどうかを分析すること。以上である。

注

1)「キータッチ・キャリア」の資料は中野靖夫氏（上越教育大学学校教育研究センター教授）より提供していただいた。

文　献

ブランド，S．　室謙二・麻生九美（訳）　1988　メディア・ラボ　福武書店
田中　敏　1994　心のプログラム　啓文社

付　記

本研究は文部省科学研究費補助金　一般研究C（No.05680172）「プログラム作成過程に視点を置いた評価法の開発」の研究成果の一部である。

Construction of Dimensions for Educational Evaluation in Computer Programming Practice

Satoshi TANAKA *

ABSTRACT

Computer programming practices has been opening a possibility that, suggested by A. Minsky, the style of learning or problem solving would shift from planner's to manipulator's. In order to evaluate the latter style, this study tried to construct some evaluational dimensions by analyzing various ways of computer programming practice. One hundred and sixty-six undergraduate students were presented two alternative ways of programming or debugging that did not change program outputs, and asked to rate their preference to either of the two ways on five-points scale. On these ratings factor analysis was conduced and four dimensions were identified as (1) avoidance to do anything new, (2) keeping clear view of descripion and contents of program, (3) makeshift and inextension policy and (4) intention to progress. Post-hoc analysis revealed that those students who had higher programming literacy had more negative loadings of Dimension 2 and Dimension 3. The results were discussed in terms of the learning style of manipulator that would be concerned with "success" of activity rather than "correctness".

* Divison of Method and Evaluation

研究発表－４

プログラミング中に発生するエピソードとそのカテゴリー化

A categorization and analysis of episode in programing process.

小岩　寿之*　　松島　健一郎*　　横田　亮宏*　　南部　昌敏**

Toshiyuki KOIWA　Kenichirou MATUSHIMA　Akihiro YOKOTA　Masatoshi NAMBU

*上越教育大学大学院

Graduate School,Joetsu University of Education

**上越教育大学学校教育研究センター

Center for Educational Research and Development,Joetsu University of Education

あらまし：プログラミングの様相を観察すると，学習者がさまざまな方法によって問題解決に取り組んでいる。プログラム作成中のキー入力データを再現できるシステムを用いて，その作成過程を明らかにしながらプログラミングにおける評価項目を抽出して，作成過程に基づく評価法を開発することを目的とした研究成果について報告する。

キーワード：プログラミング過程，情報処理言語，情報処理教育，教育測定，カテゴリー分析

1．はじめに

学校教育において行われるプログラミングは，大規模なプログラムの設計・開発とは異なり問題解決過程を直接コーディングしていくことが多い。プログラミングは与えられた問題を分析し，それを解決する方法が分かることが前提であるが，プログラミングの様相を観察してみると，学習者がさまざまな方法論によって問題解決に取り組んでいることが分かる。作成手順がきちんとまとめられ，整然とコーディングする者もいるが，逐次試行しながらコーディングする者もいる。その作成過程は，問題の解決手順が構成できたか，プログラミングに関する理解があるか，言語プロセッサに関する知識はあるか，課題に取り組む興味や関心は十分であるかなどによって左右される。すなわち，プログラミングとは多くの知識と技能を必要とする学習であり，作成者の論理展開，試行，模索等の知的ふるまいが作成過程に表出してくる。

これまで，プログラミングに関する認知の解明や教育効果を高めるなどの目的で，プログラミングに関する分析，評価研究が展開されている。宮地 (1991) はFORTRANの算術代入文について誤答分析を行い誤答原因を９つに分類している。渡辺 (1992) はPascalの学習終了後に，プログラムを読み取らせ mental running で数値計算を行わせ，その時間から理解の程度を測定している。江木ら (1990) はCOBOLプログラミング中に異常終了したリストを収集しその誤り傾向を調査している。これらの研究は，学習の終了後，あるいは異常停止が発生した時点で分析，評価しており，静的データを扱っていると言える。

一方，作成過程に着目した動的データを分析した研究も進められている。その一例として岡本ら (1992) が診断助言型のITSを用いて分析した事例があるが，この研究では，メンタルモデル解明のため机上でコーディングを行わせながら作業内容について口頭で説明させ，これをテープレコーダで録音しオンラインプロトコルを採取している。従って，実験という制約された環境下においてのデータ収集であり，かつ言語プロセッサと対話した時のありのままの作成過程ではなく技能面や記述，編集，消去，修正等の状態遷移，あるいは記述内容を変更した場合などの操作データは得られていない。前述のようにプログラミングを知的ふるまいとして捉え

れば，その作成過程のデータすなわち動的データを全て追跡し分析・評価することは，認知科学や情報教育の視点からも意義ある事と考える。プログラム作成中には，キー入力，システムの操作、修正等で，数々の出来事（エピソード）が発生する。それらのエピソードは学習の進度，学習者の理解、教授内容・方法，課題などに起因する。

本研究は，前田ら(1993)が開発したプログラミング過程を記録・再現できるシステムを用いて，測定を被験者に意識させることなくプログラム作成中のキー入力データをすべて収集し，そのデータを再び言語プロセッサに自動的に入力することにより，プログラムの作成過程を再現し，エピソードを抽出・カテゴリー化し，作成されたカテゴリーにより個人や集団の評価や分析を行うことを目的とする。

2．方法

2.1 被験者について

プログラミング過程のデータを収集する対象となった被験者は，選択科目である電子計算機概論を履修している短大１年生である。この実験までにBASICを６時間学習している。実施時期は1994年２月で，被験者人数は37名である。

2.2 課題内容

今回は以下の３つの課題についてデータの収集・分析を行った。

課題１ 底辺と高さをキーボードから入力し，三角形の面積を求めるプログラムを作成しなさい。（完成したら”TOI1.BAS”でSAVEすること）

課題２ ある旅館では宿泊料が９０００円であるが、１５人以上の団体は２０％引き，２５人以上の団体は２５％引き，３０人以上の場合は３０％引きの団体割引料金を設定している。人数を入力して宿泊費の合計 を求めるプログラムを作成しなさい。（完成したら”TOI2.BAS”でSAVEすること。）

課題３ 変数Ｎにキーボードから数値を入力し、FOR～NEXT文を用 いて１からＮまでの合計を求めるプログラムを作りなさい。（完成したら”TOI3.BAS”でSAVEすること。）

2.3 データの収集と分析方法

データの収集は，キーボード入力をシステムが逐次，キー入力バッファに保存し，キー操作の履歴データを作る。プログラミングが終了し，被験者がそのプログラムを保存するとき，ディスク（BASICの学習に使用しているもの）にキー操作の履歴データが同時に保存される。被験者にとっては，通常のプログラミングの学習を行う操作と何ら変わりなく，測定されているという意識を感じることはない。

分析手順は，まず学習が終了したデータディスクから記録されたキー操作の履歴データをシステムを用いてプリントアウトする。次にそのデータリストを見ながら，実際のプログラムの作成過程をシステムで再現し，そこで起こったできごとをメモに書き込んでいく。プログラミング作成過程で起こった消去や修正など，学習者が行った操作がすべて再現される。このとき，プログラム作成の再生速度は観察者の望む速度で行うことができる。また実際に作成されたプログラムを実行することもできる。

学習者の履歴データの一部をプリントアウトした例を図１，再現システムを用いてその履歴データを解釈した例を図２に示す。図中のリスト先頭の（ ）の数値は，その行が入力された時間（プログラミング開始からの秒数）を示している。また，BSはバックスペース・キー，CRはリターン・キー，f･5はRUN（実行），f･6はSAVE（ﾌｧｲﾙの登録）の命令実行をそれぞれ示している。

```
Data Count : 4723
  Shift key group status ···
    SHIFT: OFF
    CAPS:  ON
    ｶﾅ:    OFF
    GRPH:  OFF
    CTRL:  OFF
(0099)  １００SP’ＳＩＫＥBS BS BS BS ＴＯＩ１．ＢＡＳ
(0115)   BS BS BS BS CR
(0124)  ２００SP ＩＮＰＵＴSP”ＴＥＩＨＥＮ
(0135)   ←←←←←←←←←←←←←←←←←←←１
(1153)   →→→→→→→→→→→→→→→→→→→”；Ａ CR
(0164)  １２０SP ＩＮＰＵＴSP”ＴＡＫＡＳＡ”；Ｂ CR
(0211)  １３０SP ＩＮＰＵＴ SP”BS BS BS BS BS BS BS ＭＥＮＳ
      ＥＫＩ＝Ａ＊Ｂ／２ CR
(0249)  １４０SP ＰＲＩＮＴSP”ＭＥＮＳＥＫＩ＝”；ＭＥＮＳＥ
      ＫＩ CR
(0256)  １５０SP ＥＮＤ CR
(0257)  f･5
(0261)  １０ CR
(0268)  5 CR
(0294)  f･6ＴＯＩ１．ＢＡＳ” CR
```

図１ 課題１のキー操作履歴データ

データの時間: 4723秒
シフト・キー等の状態 ---
SHIFT: OFF
CAPS: ON
カナ: OFF
GRPH: OFF
CTRL: OFF
(0099) 100行 'SIKE を入力したが、[BS]で4文字削除して T011.BAS と入力。
(0115) .BAS の部分を[BS]で削除。
(0124) 200行を途中まで入力。
(0135) カーソル移動で行番号を 200 から 110 に変更。
(0153) 200行の続きを入力。
(0164) 120行 入力部分を入力。
(0211) 130行 INPUT を入力しかけて削除，三角形の面積の公式部分を入力。
(0249) 140行 出力文を入力。
(0256) 150行 END文を入力。
(0257) プログラムを実行。
(0261) 底辺の数値を入力。
(0268) 高さの数値を入力。
(0294) 実行結果を見て，プログラムを保存。

図2 課題1の履歴データ解釈

この事例からも明らかなように，学習者が行うBASICのプログラミング中にはさまざまなエピソードが発生する。そこで，これらのエピソードを抽出しカテゴリー化して分析を行う。

3．結果と考察

3.1 発生するエピソード

3つの課題の達成状況は以下の通りである。エピソードの抽出は，各課題ともに最後まで達成できたものについて行った。

表1 課題ごとの達成状況

課題	達成人数	割合
1	27名	73.0%
2	19名	51.4%
3	15名	40.5%
未完成	8名	21.6%

プログラミング中には，さまざまなエピソードが発生する。例えばキー入力について事例をあげると，タイピング時に2つ以上のキーを押したために（隣接キーについての誤りなど）誤って入力してしまったり，BASICの予約語・命令語や変数などの名前を誤入力したり，全く関係のないキーの入力があったりする。各課題におけるエピソードの事例を次に示す。上段は発生したエピソードの履歴データ，下段はその履歴を解釈したものである。

(0510) BS BS BS BS H A K U H I = 9 0 0 0
変数名のスペルミスを直す。
(0530) BS BS BS BS BS BS BS BS BS BS BS BS = 9 0 0 0 CR
変数名を再度変更し、続きを入力。
(0721) 1 6 0 SP P R I N T SP " S Y U K U H A K U H I - BS =
160行を入力途中にシフトキーを押さなかったため「=」が「-」になったのを変更。
(0724) " : BS ;
コロンとセミコロンのミスタッチ。
(1002) 1 2 0 SP F O R SP I = A BS 1
120行のFOR文を入力途中にミスタッチ。
(1144) ↑ ↑ ↑ ← ← ← ← K O T A E = 1 BS 0 CR
140行の変数を G から KOTAE に変更。
(1199) 1 6 0 SP N E Z BS S BS X T SP I CR
160行のNEXT文を入力途中に隣接キーのミスタッチ。

図3 各課題のエピソードの履歴データとその解釈

3.2 エピソードのカテゴリー化

プログラミングにおいて発生したさまざまなエピソードはそれぞれが，その時点での学習者のプログラムの作成行動を表している。そこで，抽出したこれらのエピソードを分類しカテゴリー化を行った。

カテゴリー化は，まずプログラム作成過程を，入力する段階（課題を解釈し，解決のための手だてを考えコーディングし入力する）とデバッグする段階（プログラム入力を終えて実行した後の作業）に大きく分けた。

分析は，すべて1行ごとの単位で行うこととし，1行の入力が終わり確定のためのリターンキーを押す前までの段階と、リターンキーを押して確定した後さらに修正を行う場合とに分けた。これは1行を入力し終わり新しく次の行を入力している途中で、すでに入力し終わった行についての修正をする場合が見られるからである。

さらに，それぞれ結果的に正しく（文法上）入力された場合と、結果的に誤りであった場合に分けた。

入力に関しては、修正がなくそのまま入力された場合と、入力途中で考え修正した場合に分けた。さらに入力中修正した場合は、キー入力（おもにキーボードの入力に関する）についてのエピソードとプログラミングについての知識や技能に関するエピソードに分けて分析した。

プログラミングについては，BASICのエディター機能に関しての知識や技能におけるエピソード，BASICでプログラミングする上での知識（命令・書式・入力方法）についてのエピソードと，問題

の解釈や解法，コーディングなど方法に関するエピソードに分けて分類した。

デバッキングの段階では、どの部分をどのように修正したかについて学習者の修正行動を分析していくことにした。まず修正の操作については，「変更」「消去」「追加」「再入力」に分け、さらに「消去」と「追加」については，部分的な修正と行単位の修正に分けた。また「再入力」は、異なった文を入力した場合と，全く同じ文を入力した場合とで分類した。学習者の作成過程の中で，全く同じ文を消したり，再び書き加えたりする行動が多く見られたからである。

修正した部分については、プログラムをその構造から大きく「入力」「処理」「出力」の三つの部分に分け，さらにそれらを命令（BASICの命令）・書式［命令］（命令文の書式）・書式［BASIC］（命令文以外のBASICの書式）・変数・文字列・計算式に関する修正に分けて分析した。

3.3カテゴリー別発生度数

プログラムの入力段階において発生したエピソードを，カテゴリーごと分類したものを表1に示す。

3.4カテゴリー化によるプログラミングの分析・評価

3.4-1個人の時系列データ

カテゴリー分析では時系列分析が可能である。ある学習者の課題3のプログラム作成過程を分析した。入力時の特徴としては、行を入力している途中で（まだ確定をする前に）前の行にカーソルを移動して変更することがしばしばみられた。また、READ命令を入力しているのであるが、その行動の様子を解釈すると、READ命令の意味を理解した上で行っていないことが明らかであった。これは操作履歴から読みとったものである。

デバッグ時の特徴を明らかにするために、一連の行動をデバッグ時の変更行動カテゴリーを用いて分析した。プログラムが完成するまでに７９回の変更行動を行っていた。図４に変更行動の度数分布を示す。変数の変更が特に多いことがわかる。図５に、デバッグ過程の流れを示す。（図中の記号Ｒは、プログラムの実行、Ｉは、データのキーボード入力、Ｌは、プログラムリストの画面表示、Ｍは、命令の変更、Ｓｍは、命令の書式変更、Ｓｂは、BASICの書式変更、Ｈは、変数の変更、Ｊは、文字列の変更、Ｋは、計算式の変更、Ｇ１は、消去、Ｇ２は、追加、Ｒ１は、異なる文の再入力、Ｒ２は、同じ文の再入力、Ｘは、その他の行動を示す。）

全体を通して言えることとしては、変数の変更をしてみては実行し、期待した値が出てこないとわかると、また変数を変更して実行することを繰り返していた。一連の変数の変更の様子からはその意図が何であるかを解釈することができない場面がほとんどであった。プログラム全体の流れの中での変数のもつ役割として、どの変数に何を割り当てているのかということを理解していないと考えることができる。また、ループ文を使用する際の変数の変化（ループの回数をカウントする変数の変化）を理解していないのではないかと考えられる。さらに、どの変数に何の値を代入しているかを把握していないため、求める値を計算する場合に、どの変数を使うかで悩んでいるようすがうかがえた。

3.4-2集団のデータ

発生したエピソードをカテゴリー化し，カテゴリーごとの発生度数をグラフ化したものを次に示す。

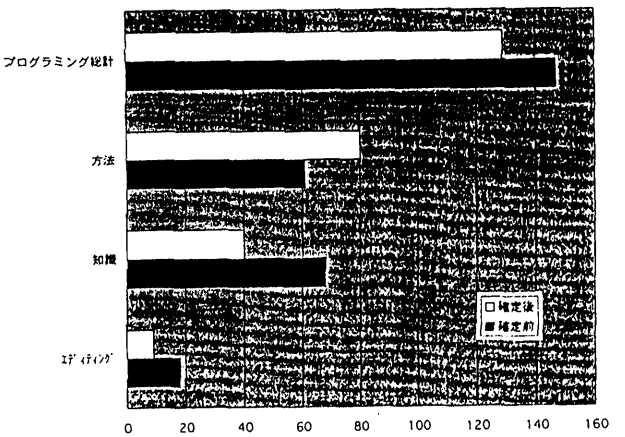

図７ プログラミングに関する修正度数

図７はプログラミングに関する修正度数を表したものである。全体としてプログラムの方法や知識に関する修正が多く見られる。今回のデータからは，プログラムを入力し確定前に行われる修正が，どの過程でも多く見られるが，プログラムの方法に関する修正では，確定後に行われる修正が多いという結果が得られた。これは学習者が，問題解決のための全体的な見通しが立てられず，入力しては修正するという場当たり的な解決方法でプログラムの作成を行っていたことが考えられる。

表2　発生したエピソードとそのカテゴリーごとの発生度数（プログラム入力段階における）

カテゴリー				エピソード	発生度数			
修正なし					250	250		865
入力中の修正	キー入力	ミスタッチ		タイピング時に2つ以上のキーを押してたりして，誤って入力された場合。 Basicの予約語・命令語以外の変数などの誤入力の場合。	215			
		隣接キーの間違い		ミスタッチの中でキーボード上の隣あったキーを間違えて入力した場合，特にキーボードに慣れていない初心者に多くみられる。	96			
		不要入力		上記以外で全く関係のないキー入力があった場合。	28			
	プログラミング	エディティング	消去方法	BS･DEL･SPACEキーなどの操作ミスで、消去するべき所を消去できなかったり，消去しなくてもよい所を消去してしまった場合。	12	27	218	
			再入力・追加方法	追加入力する際に，操作を誤って追加できなかったり、余分な追加をしてしまった場合。	10			
			カーソル・スクロール機能	カーソルキーの使い方やスクロール操作を誤った場合。	5			
		知識	予約語等のスペリング・ミス	BASICの予約語や命令語などのスペルを間違えた場合。	25	108		
			文の構造・書式	一文の中での命令の構造や，BASICの予約語，命令語の書式を間違えた場合。	69			
			コマンド・ミス	BASIC上のコマンドの間違い。（例：注釈文に'をつけない。）	14			
		方法	問題の理解・解釈	与えられた問題そのものを理解していない場合。	15	115		
			問題解決の方法	与えられた問題は理解しているが，その解決方法が分かっていない場合。	18			
			コーディング・表記	問題の解決方法を表した計算式やプログラムが間違っている場合。	68			
			プログラムの構造	プログラムの流れが間違っている場合。	40			

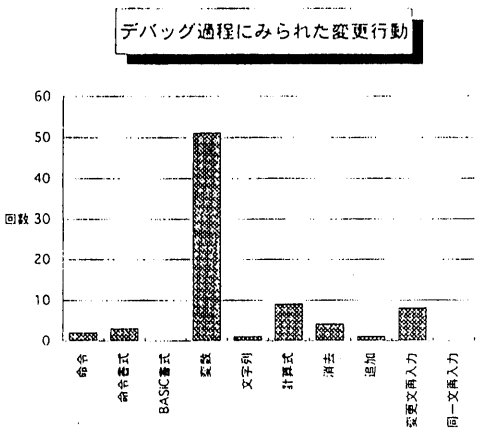

図4　変更行動の度数分布

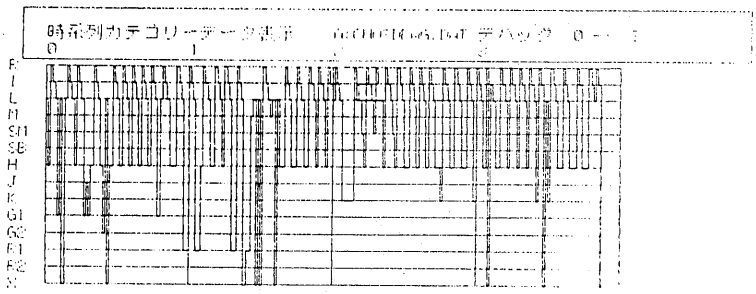

図5　時系列カテゴリーデータ

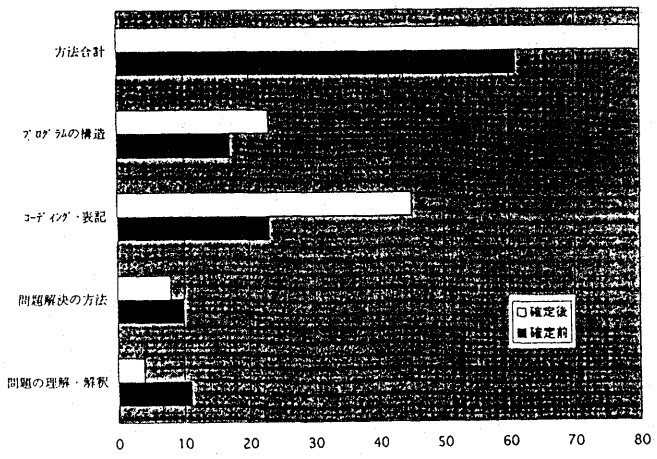

図8 プログラミングの方法についての修正度数

図8はプログラミングの方法についての修正度数を表したものである。問題に対する理解や解決方法に関することよりもコーディング・表記やプログラムの構造（流れ）に関する修正が多く見られる。これは課題解決よりもBASICについての基礎的な知識の不足やプログラム作成にまだ慣れていないことが考えられる。

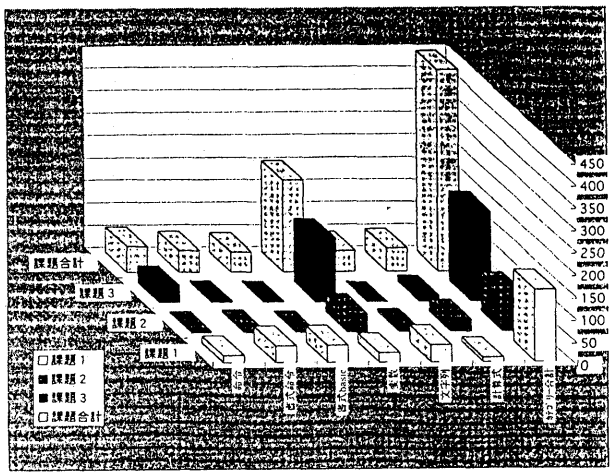

図9 課題ごとのデバッグにおける修正度数

図9はデバッグにおける修正度数をカテゴリーごとに表したグラフである。課題合計で見ると変数についての修正が多いことが分かる。これは個人の時系列データの分析と同じような結果である。全体的にプログラム中の変数の理解が不十分であることが分かる。課題ごとに修正度数を見ると課題3が最も多く，次いで1，2の順になっている。課題3では変数と命令の修正が多いことから，FOR～NEXT文の使い方が難しかったことが考えられる。難易度から考えると課題1と2の修正度数が逆転している。課題1で多かった命令文の書式やBASICの書式についての修正が，課題2及び3では減少している。修正が減少したのは，課題1においてBASICの基本的な命令や書式が学習できたためと考えられる。

4．おわりに

プログラム作成は，机間巡視やコンピュータネットワークのモニタリングなどにより観察することができる。しかし，教師は全員の作成過程をすべて把握できるわけではない。本研究の方法によれば，多人数の作成過程を分析者が望む速度で分析できる。作成過程は学習者の知的な振る舞いが表出しており，通常のプログラムリストの提出では見ることのできない過程を観察することができる。評価問題を設定し，作成過程を分析すれば学習者，教育内容，システムなどの評価が可能になると考える。

本研究は文部省科学研究費補助金一般研究C (No. 05680172) 「プログラム作成過程に視点を置いた評価法の開発」の研究成果の一部である。

参考文献

江木鶴子，岡村健史郎，長田一興（1989） ＣＯＢＯＬプログラミングにおける初心者の誤り傾向．ＣＡＩ学会誌，6（3）：26-36

江木鶴子，岡村健史郎，長田一興（1990） ゴール／プラン分析法による初心者の作成したＣＯＢＬプログラムの誤り分析．ＣＡＩ学会誌，7（2）：80-91

岡本敏雄，安田恭一郎（1992） Ｃ言語プログラミング過程のメンタルモデルの分析－診断助言型のITSを用いて．日本教育工学会雑誌，16（3）：119-130

カーニハン，リッチー著，石田晴久訳（1989） プグラミング言語Ｃ 第2版．共立出版

前田恵三，中野靖夫（1993） コンピュータ操作過程の再現システム．日本教育工学雑誌，16（4）：185-195

前田恵三，中野靖夫（1994） Ｃ言語のプログラミング過程の分析．電子情報通信学会技術研究報告，ET94-66：37-44

宮地功（1991） ＢＡＳＩＣ入門教育における学習者の変容について．ＣＡＩ学会誌，8（2）：90-98

渡辺馨，渡辺祥子，神宮英夫（1992） プログラミング能力の測度としての反応時間．ＣＡＩ学会誌，9（1）：33-37

口頭発表－１

Logo学習におけるキー操作の分析

An analysis of key-operation on Logo learning

近藤　智嗣
KONDO,Tomotsugu
株式会社 新 学 社
SHINGAKUSHA CO.,LTD.

中野　靖夫
NAKANO,Yasuo
上越教育大学
Joetsu University of Education

要　約　プログラミングの学習過程で，学習者のつまずきを指導者が把握すれば指導や教材にフィードバックすることが可能となる．そこで，Logo言語でプログラミングを学習する場合の操作過程を明らかにするため，キー操作を記録し分析した．

ｷｰﾜｰﾄﾞ　情報教育，Ｌｏｇｏ，中学教育，情報基礎，学習過程，学習履歴

１．はじめに

本年度より「情報基礎」の授業が開始されたが，いくつかの問題が残されたままとなっている．例えば，プログラミング言語としてBASICやLogoが使用されているが，その指導内容，指導方法は確立されておらず暗中模索の状況と言える．プログラミングのように確実に手順を追わなければならない授業では，学習者は一度つまずいてしまうと，その先に進むのは困難である．

そこで本研究では，そのつまずきの箇所を調べ，情報基礎におけるプログラミングの指導法や教材制作の基礎資料とするため，Logo学習のキー操作を記録し分析することにした．今回は学習段階の違う2つの授業について試みたのでここに報告する．

２．方法

Logoのシステムにキー操作記録プログラムを常駐させ，自動的に履歴をとれるようにした．調査は「技術・家庭科」の時間に行い，普段と同じように授業を進めた．

(1)対象

東京都内の公立中学校３年生

◇Ａ中学校：30名(15台)

Logoは最初の授業．以前にBASICを経験．

◇Ｂ中学校：37名(20台)

Logoは6回目の授業．

※2校とも2人で1台のパソコンを使用しているため，データ数は台数分である．

(2)調査日

◇Ａ中学校：1992年12月11日

◇Ｂ中学校：1993年 7月12日

(3)使用機器

NEC PC-9801シリーズ

(4)使用ソフトウェア

・BANDIT LOGO Ver.1.97(ﾌﾘｰｿﾌﾄｳｪｱ)

・前田・中野(1992)によって開発されたキー操作記録プログラム

(5)分析の方法

分析をする単位は，入力画面に命令を入力した場合は[ﾘﾀｰﾝ]を押すまで，編集画面にプログラムを入力した場合はプログラムを実行するまでとした．

分析の視点は，キー操作の特徴，命令のスペル間違い，命令の書式間違い，制御構造の間違い，内容の理解を中心に行った．

(6)授業の概要

◇Ａ中学校は，命令を入力画面に直接入力．内容は，forwardとrightの意味．repeatを使って正方形，正三角形，円をかいた．

◇Ｂ中学校は，編集画面にプログラムを書き，入力画面で実行した．

内容は，正方形を繰り返し回転させた模様をかき，回転させる角度を変えた．

３．結果の一部と考察

◇Ａ中学校

○キー操作の特徴

(1)[DEL]を使わず[→][BS]	73%
(2)[BS]を使わず[←][DEL]	20%
(3)括弧[]の入力ミス	67%
(4)不要な[INS]	60%
(5)空白の入力で[→]	33%

(1)(2)は文字を消去する場合[BS]か[DEL]のどらか一方のみを使い[→]か[←]と併用するため操作回数が増えてしまっている．両方で9割以上にもなる．

(3)repeat文の後の括弧[]の入力で {} ()<> 等入力してしまったものが7割弱．

(4)BANDIT LOGOは[INS]に機能はないが，6割が不必要な箇所で数回使用している．これは以前に学習したBASICの影響と思われる．

(5)空白の入力で[ｽﾍﾟｰｽ]を押さず，[→]でｶｰｿﾙを移動させようとしたものが約3割．

○課題1　forward 100と入力．

(1)スペルの間違い	33%
(2)forwardのみで[ﾘﾀｰﾝ]	47%

(1)スペルには以下の間違いがあった．

forweard, forard, foward, fowerd, forwerd

(2)引数を入力せず，forwardのみで[ﾘﾀｰﾝ]を押したものは約半数．

この課題を入力するためには最低12回キーを押す必要があるが，これができたのはわずか2人であり，570回もキーを押し，修正を繰り返したものもいた．

○課題2　repeat文を使って正三角形をかく．

(1)角度を60にした	47%
(2)その他の角度	33%
(3)repeat文を使わない	20%

最初は全員が失敗．中でもタートルを回転させる角度を60度としたものが約半数で，最終的に完成できないものもいた．

◇B中学校

○キー操作の特徴

6回目の授業でこのLogoにも慣れてきているためか主だった誤入力はなかった．その他には，[TAB]で字下げするよう指示していたが，それがくずれても直すものはいなかった．プログラムの構造を見やすくするために字下げをしているという認識が無いように思われる．

○課題1　正方形とそれを回転させる2つの命令を作り，回転させる角度を変えてみる．

2つの命令は黒板に提示し入力させた．

(1):(ｺﾛﾝ)→;(ｾﾐｺﾛﾝ)のﾐｽ	10%
(2)正方形の角度を変えた	20%

(1)入力ミスとして引数の先頭の:(ｺﾛﾝ)を;(ｾﾐｺﾛﾝ)と間違えたのが1割．

(2)回転させる角度の変更の際，間違って正方形の命令を変更したものが2割．2つの命令の意味が理解できてないことが考えられる．

4．まとめ

今回の調査で，学習の初期においては学習内容よりもむしろキー操作に不慣れな面でのミスが予想以上に多く，効率の悪いキー操作をしていることがわかった．授業1時間でのキーを打つ総数をみても，以下のように多いことがわかる．

	平均	最大	最少
A中学校	1694	3320	457
B中学校	1942	6584	614

これは，初学者のキー操作の特徴として，思うように入力できない場合，試行錯誤で修正をしているためと考えられる．このため，プログラムのアルゴリズムを理解する以前に与えられたリストを正確に入力することが生徒の課題となってしまっていると言えよう．

また，キーを打つ回数が必要以上に多い理由の一つとして次の場合がある．エラーが生じた時，入力画面にもエラーメッセージが表示されるが，6割以上がそれを消すために[BS]を多用していた．これは，今回使用したLogoの機能に起因するものである．

その他にプログラムの制御構造に関しては，今回の2つの授業からは特に間違いなどは見られなかった．

5．おわりに

生徒のつまずく箇所は学習の各段階で異なるため，今後は一連のカリキュラムの中で継続的な調査を進めたい．また，それを踏まえた上で指導方法や教材にフィードバックしなければならないと考えている．

さらに，この調査はソフトのユーザーインターフェースの評価の一方法としても有効である．ソフト開発のプロトタイプの段階で行うことにより，操作を間違えやすいユーザーインターフェースを避けた設計に改善することも可能だと考えている．

【参考文献】

前田・中野(1992)，コンピュータ操作の記録・再現，日本教育工学会研究報告集　JET 92-4

口頭発表－２

Ｌｏｇｏプログラミングのカテゴリー分析

An analysis of programming process in the language Logo

近藤智嗣
KONDO,Tomotsugu
文京区教育センター
Bunkyo Ward Center for Education

中野靖夫
NAKANO,Yasuo
上越教育大学
Joetsu University of Education

〈あらまし〉 中学校の情報基礎におけるプログラミングの学習過程の実態を明らかにするために，キー入力データを記録するプログラムを使用し，Logo言語の学習履歴をカテゴリー分析した。

〈キーワード〉 情報教育，Logo，中学校教育，情報基礎，学習過程

１．はじめに

中学校の情報基礎にプログラミングが導入されるようになり，その使用言語にはLogoも多く利用されている。

しかし，成果としてのプログラムは評価されているが，知識生成としてのプログラム作成過程の評価は経験的な机間巡視程度にとどまっており，あまり，分析されていない。

そこで，本研究ではプログラミング行動の作成過程をカテゴリー分析し，理解，作成の状況を明らかにすることにした。

２．実験方法

1)対象

山形県の公立中学校2年生24名

2)調査日

1994年2月26日

調査は「選択技術」の時間に行った。

3)学習形態

2名1組で1台のパソコンを使用したものと，1名で1台のパソコンを使用したものがおり，データ数は17である。

4)学習経験

Logo学習は10時間目

5)使用機器

・NEC PC-9801
・LogoWriter2
・キー操作の再現システム

6)データ収集法と処理法

再現システム（日本教育工学雑誌 16(4) 185-195参照）を使用し，カテゴリーで分析した。

7)評価問題

課題①タートルを一方向に動かす
・タートルの色，形，位置を決める手続きを作る
・タートルを動かす手続きを作る
課題②動きの速さを変える
・命令を適切な場所に追加する
課題③タートルの色，形を変えないでいろいろな動きをさせる
課題④タートルの色，形を変えていろいろな動きをさせる

課題①については，以下に示すプログラムを参考にして評価問題をさせた。

```
TO KAME
 SETC 3 SETSH 2
 PU
 SETPOS [50 50]
 SETH 90
END
```

```
TO UGOKU1
 REPEAT 50 [FD 3]
END
```

図１ 課題①の完成プログラム例

課題②～④については，命令のみ説明し，生徒には自由に作成させた。

3．結果の一部と考察

1)課題の達成度

課題①は17で全員，課題②は14，課題③は12，課題④は4が達成した。

2)カテゴリーの抽出

プログラミングは，エディタにプログラムを入力し，実行した結果，不具合があれば修正する。そしてさらに加工しながら完成させていくという過程をとる。この過程の中では，プログラムの構造を考える論理的なことと，タイピングなどの技能的なことが必要になる。ここではその二つの要因ごとに分析した。

まず，カテゴリーは入力，修正，加工に分け，それらの正誤，誤りの箇所で分類した。入力は結果として論理的に正しい場合でも，そのまま入力した場合と，途中で修正しながら入力した場合に分けた。修正は実行後に，誤りを直した場合である。

3)課題①の分析結果（論理面）

ここでは，全員が到達した課題①に関してカテゴリー分析した。表1がそのカテゴリーと結果である。件数は17個のデータの出現件数の合計である。

表1　プログラミングのカテゴリー

	正誤	状況	誤りの箇所	件数
入力	正	修正なし		92
		入力途中での修正	書式	8
			命令	28
			数	18
			文字記号	8
			空白挿入	7
	誤		書式	2
			命令	3
			数	
			文字記号	
			空白挿入	6
修正	正	実行後デバッグ	書式	
			命令	3
			数	
			文字記号	2
			空白挿入	
			空白削除	4
	誤		書式	
			命令	
			数	
			文字記号	
			空白挿入	
			空白削除	2
加工	正		数値変更	12
			命令変更	2
			命令追加	3

間違いの多くは実行する前に入力途中で誤りに気づき修正されていることがわかる。中でも，命令のスペルミス，引数となる数を変更しているものが多い。また，命令と命令，命令と引数の間には空白を入れて区切りとするが，空白を入れ忘れるミスが7件あった。

逆に空白を入れてはいけないところに空白を入れてしまったミスは6件あり，これは実行後エラーが発生してから修正されている。その他には，見かけ上は正しく見えても，前の行で改行されておらず2行がつながってしまってエラーとなったものもいた。命令のミスよりも空白の有無によるミスなどの方が初学者にとっては気づきにくいと言えるだろう。

また，誤りではないが，行や文字を挿入する場合，挿入する前におおよその文字数分空白や改行を入れたり，正しく入力されている命令を一旦消し，挿入後再入力するという例も少なくなかった。エディターの挿入の概念が把握されていないための不要な動作である。

4)キー入力の分析結果（技能面）

使用したLogo言語は大文字小文字の区別がないため，SHIFTｷｰは記号の入力時にのみ必要となる。そこで，記号の入力ミスを見てみると，［"］が入力できないものが8，［.］が入力できないものも8であり，約半数であった。また，ファイル名や手続き名に仮名を使用しているのは17のデータ中14で，その内，何らかのミスをしたものは13とほとんどがｶﾅｷｰの操作時にタイプミスをしていた。課題①～④での総数は38件に及んでいた。

4．まとめ

プログラミングの過程をカテゴリーにして分析した結果，プログラミングの初歩の段階では，論理的な誤りよりも，むしろ，プログラムの約束ごとである，区切り文字やキー操作時のｶﾅｷｰやｼﾌﾄｷｰによるミスが課題解決に影響を及ぼしていることがわかった。

なお本研究は文部省科学研究費補助金，一般研究C（05680172）"プログラム作成過程に視点をおいた評価法の開発"による研究成果の一部である。