プログラミング初心者を対象にした
デバッグ戦術の学習支援に関する研究

江　木　鶴　子

目　次

図一覧

表一覧

1. はじめに

本研究の主題はプログラミング教育の支援に関するものである。本論文では、プログラミング教育の初期段階でプログラム構築技術のひとつであるトレースを指導する支援システムを提案する。さらに提案した支援システムを実際の教育で使用し、その結果を評価すると共にトレース行動がプログラミング学習に与える影響について論じる。

プログラミング教育では、教育の初期段階からプログラミング言語、アルゴリズム、プログラム構築技術など、分野の異なる要素を平行して指導する。学習者は，図 1.1 に示すような個々の分野の項目を与えられた学習環境で統合化しながらプログラミングを学ぶ。多くの学習者はプログラミング言語の文法やアルゴリズムなどの各分野の教授内容を理解できていても，これらをプログラムとして統合化する構築段階で行き詰まることが多い．プログラムのデバッグ時に行き詰まりを解消するために必要となる知識は多種多様であるが、これらをすべて事前に教授することは不可能である。そのため，学習者はデバッグ段階でプログラムの誤りを契機に様々な試みをする．その過程で，プログラミング言語やアルゴリズムだけでなくプログラム作成に関する幅広い知識を習得する．このデバッグ過程が学習者のプログラム理解を深めプログラミングに関する新しい知識発見や定着のための場となる [1)]．したがって学習者はこのデバッグ過程を通してできるだけ多くの体験をすることがプログラミング学習に繋がる。これが初期のプログラミング教育において演習や実習が必須の教育課程と位置づけられている理由である。本論の支援目的は，このデバッグ過程で構築技術のひとつであるトレースを指導することによってトレース技術を獲得させるとともにデバッグを促進させ，これを手掛かりに学習者自らが学ぶ機会を意図的に増加させ,学習者が能動的にプログラミング学習を進めることができるようにすることである．

本論では，次章の「プログラミング学習支援に関する関連研究」に続き，プログラミングの学習者が教授されたプログラミング言語やアルゴリズムの知識を用いてどのようにプログラムを作成していくかの過程を詳細に分析し，デバッグ時の学習者の行動を明らかにする（「3　学習者のプログラム生成過程の分析」）．この分析で，学習者は課題プログラムに必要なプログラミング言語やアルゴリズム知識だけでなく，プログラミングを進めるための多様な戦術を使用しており，それがプログラミング学習に影響を与えていることを示す．次に，これらの戦術を学習者がプログラミングの初期段階で如何にして獲得しているかを調査し，その結果として，学習者は幾つかの戦術に関しては教師から何の指導も受けないまま自らが技術獲得し使用するが，トレースに関しては指導が必要であることを明ら

かにする（「4　プログラミング戦術学習に関する分析」）．さらにトレースに関する指導は，学習者の状況に合った個別の指導が必要であることも示す（「5　トレース支援システムの必要性」）．以上の調査分析を基にトレースを指導するデバッグ支援システム（Debugging Support System を略して DESUS と呼ぶ）を構築した．「6　トレースを指導するデバッグ支援システム」で DESUS の詳細について述べる．最後に構築した支援システム DESUS を実際の教育で使用した結果を分析し，支援システムの評価とトレース学習がプログラミング学習に与える影響について述べる（「7　DESUS を用いたプログラミング教育の実験と評価」）．

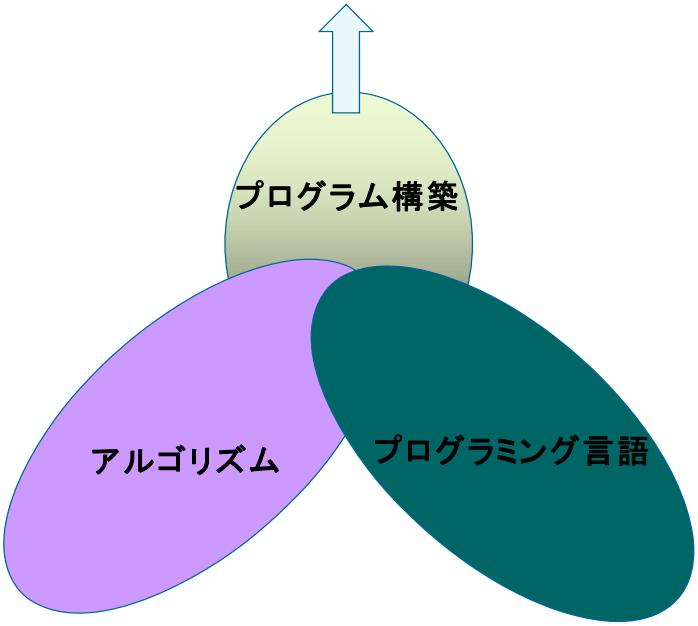


図 1.1　プログラミング教育の要素

2. プログラミング学習支援に関する関連研究

初期のプログラミング学習者には，図 1.1 に示す教育要素を明確に分別できないことが多い．特にプログラムデバッグ時に起こる誤り現象からそれを正確に判別することは困難である．そのため作成したプログラムが意図したとおりに動かないことをきっかけに，学習者の試行錯誤が始まる．プログラミング教育の支援システムに関する研究は，この段階で学習者に図 1.1 に示す教育要素のどこかに焦点をあてて支援するものが多い．

アルゴリズム理解を主たる目的とする支援システムが比較的多く，これらは大きく分けて 2 通りある．ひとつは，システムが学習者のプログラムを診断しこの診断をもとにプログラムの誤りを指摘し解決策を支援するものであり，プログラムの誤り同定方法の違いによりいくつかのシステム[2),3),4),5),6),7,8)]が提案されている．もうひとつは学習者のプログラムを別の形に表現することにより学習者に誤りを気づかせるという方法で支援するシステム[9),10),11),12),13),(14)]である．前者は学習者プログラムの誤り同定をする機構の開発が大きな比重を占め，特定のアルゴリズムや特定のプログラミング言語に特化したシステムとして開発されることが多く，プログラミング教育で実践的に利用される支援システムとして使用するまでには至っていない．それに対して後者の支援は，プログラムを表現する方法により多様なシステムが報告されている．この方法ではプログラムの誤りを学習者に直接指摘するのではなく，誤りを学習者に気づかせるための表現方法に工夫がある．

プログラミング言語への支援は，プログラム翻訳段階でのエラーについて学習者に分かりやすい誤り指摘をするもの[15),16)]，プログラム入力段階で正しい構文を入力できるように支援をするもの[17),18)]，また初心者に分かりやすい教育用プログラミング言語の提案なども報告されている[18,19)]．

そのほか，学習者の学習段階に応じた課題を提示するシステム[20,21)]や，デバッグ過程で起こした誤りを記録させる機能を開発環境に設け，学習者に誤りを記載させることにより内省させ，学習者自身が起こした誤りから学ばせようとする支援も行なわれている[22)]．また，教師を支援する学習者プログラムの評価システムや提出されるレポートの授受管理システムなど，プログラミング教育を側面から支援するシステムなども報告されている[23),24),25),26)]．

それに対して，プログラム構築技術に関する支援は，eclips や開発環境独自のデバッガーやトレーサーなどとして実際に実用化されたシステムは多いが，ほとんどのシステムは支援対象がエキスパートでありプログラム開発の効率が重視された開発支援システムであ

る [27]．エキスパートを対象にした開発環境では，トレースなどの技法についての概念理解ができていることを前提にして支援が行なわれており，技法についての理解ができていない初心者がそれを使いこなすのは難しい．プログラミング初心者の開発環境には，開発の効率化よりも開発を行うための様々な技法が学習できる支援が必要である．現在使われているプログラム開発環境には，初心者が構築技術やデバッグ方法を学習するための支援機能は備えられていない．

本論で提案する DESUS は，デバッグの行き詰まり状態でトレース技法を指導する機能を備えたプログラミング開発環境である．トレースというデバッグ時に使用する技術指導という意味で構築技術の支援に関するシステムに分類されるが，トレースを指導することによって学習者のプログラムを別の目的で実行させるという点では，プログラムを別の形に表現して学習者に誤りを発見させようとする支援システムとも言える． DESUS の指導の特徴は，トレース指示に関しては支援を求めた学習者のプログラムを用いて具体的にするが，トレース実行に関してはシステムが実行して見せる，あるいはシステムが実行した結果の情報を提供するなどシステム主導の支援方法ではなく，学習者自身が DESUS の指示に基づいてトレースすることを期待する間接的な指導方法をとることである．学習者がトレース技術を獲得するためには，学習者自身がトレースを試みる行動が重要であるとの考えからである．支援によりトレース技術を早期に獲得させることにより，学習者の自立的なプログラミング学習を促進させることが目的である．

3. 学習者のプログラム生成過程の分析

3.1 プログラム生成過程分析の目的

プログラミング教育支援システムを構築するためには，支援が必要な箇所の特定と，どのように支援するかの 2 つの側面から，学習者のプログラミング学習に関する事前分析は欠かせない．先見的研究の多くは，次の 2 通りの分析が行われている．

1） 学習者プログラムの誤り分析

2） プログラム生成過程の分析

上記１）の分析は，学習者のプログラムに現れる誤り現象を様々な角度から分析し学習者の誤概念や知識構造を究明しようとする試みである [28),29,30)]．２）の分析は，プログラム生成過程を学習者の知識獲得過程や問題解決過程と捉え，指導方略の解明や学習のメカニズムを明らかにしようとする試みである [31),32),33),34),35),36)]．本研究では，当初１）の誤り分析 [37),38),39)]を，次にデバッグ支援としてどのような支援を実施するかを明確にする目的で，２）の学習者のプログラム作成過程の分析を行った [40),41),42)]．DESUS のデバッグ支援は，主に２）の分析結果を基に構築された．そこで本論では，２）の分析からデバッグ時の行き詰まり状態で学習者がどのような振る舞いをするかを述べ，トレースなどのデバッグ上の戦術がプログラミング学習に効果的に働いていることを明らかにする．

3.2 調査対象

3.2.1 被験者

被験者としてプログラム作成を行った学習者は，山口大学工学部電子工学系の学科に所属する学生 4 名（sm，kt，yk，fm とする）と宇部短期大学情報計数学科（現　宇部フロンティア大学短期大学部）に所属する 2 名（nt，yt とする）の合わせて 6 名である．学習者は C 言語を用いてプログラムを作成した．

4 名の大学生は，いずれも教養課程で計算機概論を受講し，その後半期間のプログラミング教育を FORTRAN で受けていた．4 名のうち sm と kt は，卒論で C 言語が必要なため調査時に独学で C 言語を学習中であった．yk と fm は，大学入学以前には全くプログラミングの経験はなく，大学で初めて FROTRAN でプログラミング教育を受けたが，一人で作成することに多少不安を持っていた．卒論では AWK を使用し，C 言語に関しては本で少し見たことがある程度であった．

被験者 nt と yt は，1 年前期から C 言語によりプログラミング教育を受けており，観察時は 1 年後期後半であったので約半年間のプログラミング教育を受けた段階の学習者である．

上記被験者のうち，nt,と yt は，受講している演習科目の課題としてプログラムを作成した．その他の sm，kt，yk，fm は，この観察実験のために特別にプログラムを作成した．以上のように観察時の学習者は，すでに何らかのプログラム経験を持っており，最も初期段階のプログラミング教育は終了した段階の学習者である．被験者の概要を表 3.1 にまとめる．

表 3.1 プログラム生成過程分析の被験者

被験者名	所属	プログラム経験	課題作成の目的
nt	短期大学，情報系学科	半期間 C 言語	演習課題として
yt	短期大学，情報系学科	半期間 C 言語	演習課題として
sm	大学工学部，電子情報工学系	半期間 FORTRAN C 言語独学中	観察実験のため
kt	大学工学部，電子情報工学系	半期間 FORTRAN C 言語独学中	観察実験のため
yk	大学工学部，電子情報工学系	半期間 FORTRAN AWK（卒研で使用）	観察実験のため
fm	大学工学部，電子情報工学系	半期間 FORTRAN AWK（卒研で使用）	観察実験のため

3.2.2　調査対象課題

観察の対象とした課題は，宇部短期大学の情報計数学科で 1 年後期のプログラミング演習で課される C 言語の 13 個の課題である．その課題内容と，各被験者が作成した課題を表 3.2 に示す．各被験者は，表中の○印がついた課題を表の上から順に作成した．

被験者 nt と yt は，課題２a を除く 12 課題を作成した．被験者 sm，kt，yk，fm は，課題１，課題２a，課題３，課題４，課題 12，課題 17 の 6 課題を作成した．全員が作成した課題のうち，最後の課題である課題 17 を分析対象課題とした．

課題 17 の処理内容は，ある映画館の 1 ヶ月間の入場者ファイル（日付，曜日，映画名，一般入場者数，中高入場者数，小学入場者数を 1 レコードとして編成された順ファイル）を入力データとして，映画名別の入場者数を種類別（大人，中高，小学）に集計表を出力し，最後に 1 ヶ月間の入場者総合計を表示するものである．

表 3.2　プログラム生成過程分析の被験者が作成した課題

課題番号	課題内容	被験者	
		nt, yt	sm,kt,yk,fm
課題１	条件文，合計，表示	○	○
課題２a	平均，分類		○
課題２b	平均，最大値	○	
課題３	ファイル操作（出力）	○	○
課題４	ファイル操作（入力）	○	○
課題 11	データファイルの作成	○	
課題 12	一覧表示	○	○
課題 13	集計，表示	○	
課題 14	関数	○	
課題 15	集計，アナログ表示	○	
課題 16	集計，アナログ表示	○	
課題 17	グループ集計，表示	○	○
課題 18	配列，集計，表示	○	

3.3　分析方法

3.3.1　データとその役割

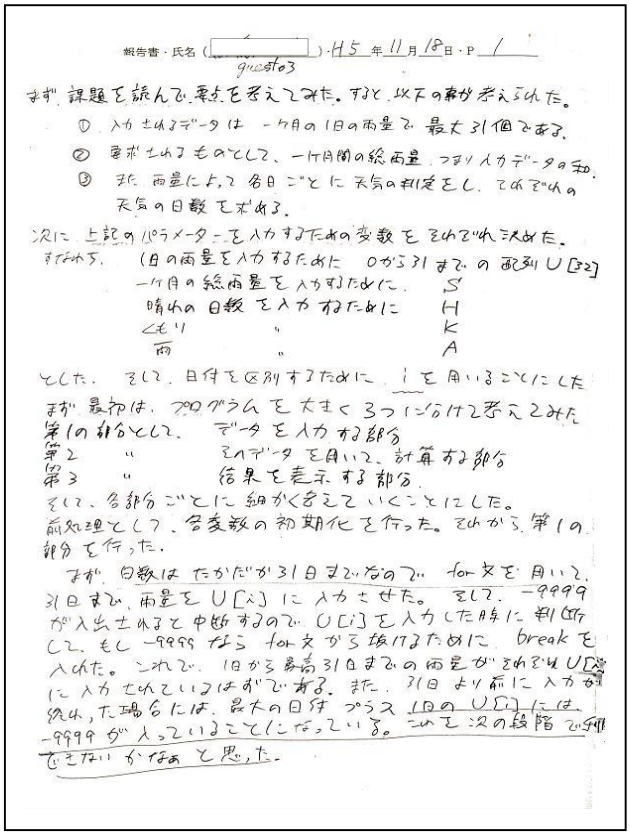
プログラム生成過程の分析のために 2 通りの観察データを取得した．ひとつは，学習者のプログラム生成過程時に翻訳したすべてのプログラムである．これは，学習者がプログラム構築をどのような過程を得て完成させたかの客観的な行動データである．それに対してもうひとつのデータは，客観的なデータであるプログラム生成過程プロセスを基に学習者自身にその行動を内観させ発話したものを記録したものである．本章の調査では，主としてこれら２つの観察データを基にプログラム生成過程調書を作成し，プログラム生成過程モデル構築に利用した．以下に各資料の詳細と役割を述べる．

（1）　オンラインプロトコル

学習者によってデバッグ時に翻訳されるプログラムをすべて保存したデータを本章ではオンラインプロトコルと呼ぶ．これらのプログラムを時系列に並べ，その差を取ると学習者のプログラム生成過程行動データが作成できる．このデータの収集は，UNIX の RCS 機能と TurboC のファイルセーブ機能を利用して収集した．

（2）　作業報告書

1 回の作業あるいは 1 課題の終了時に，その間のデバッグ過程を振り返り，その内容を学習者が報告書として自由に記述したものである．これは自己申告書で学習者自身による作成の意図や作成過程の覚書とも言える．これは，後に行われるプログラム作成過程を振り返って内観する際の重要な手掛かりとなる資料である．図 3.1 に作業報告書の実例（一部）を示す．


報告書・氏名（guest03）・H5 年 11 月 18 日・P 1

まず、課題を読んで考えてみた。すると、以下の事が考えられた。

① 入力するデータは 一ヶ月の1日の雨量で 最大31個である。

② 要求するものとして、一ヶ月間の総雨量（つまり入力データの和）

③ また 雨量によって 各日ごとに 天気の判定をし、でのでの天気の日数を求める。

次に 上記のパラメーターを入力するための変数をそれぞれ決めた。

すなわち、（日の雨量を入力するために 0から31までの配列U[32]

一ヶ月の総雨量を入力するために S

晴れの日数を入力するために H

くもり 〃 K

雨 〃 A

とした。そして、日付を区別するために i を用いることにした

まず最初は、プログラムを大まかに3つに分けて考えてみた

第1の部分として、データを入力する部分

第2 〃 そのデータを用いて、計算する部分

第3 〃 結果を表示する部分

そして、各部分ごとに細かく考えていくことにした。

前処理として、各変数の初期化を行った。それから 第1の部分を行った。

まず、日数はたかだか31日までなので、for文を用いて、31日まで、雨量を U[i] に入力させた。そして、-9999 が入力されると中断するので U[i] を入力した時に判断して、もし -9999 なら for文から抜けるために break を入れた。これで、1日から最高31日までの雨量がそれぞれ U[i] に入力されているはずである。また、31日より前に入力が終了した場合には、最大の日数プラス 1日の U[i] には -9999 が入っていることになっている。これを次の段階で利用できないかなあと思った。

図 3.1　作業報告書の実例（一部）

（3）　発話プロトコル

観察者が，学習者に（1）と（2）のデータを示して，プログラム作成時の意図などをインタビューし，それらを忠実に記録したものである．このデータは，オンラインプロトコルに記録された作成過程を読み取る際の手掛かりとして重要であり，観察者の主観だけによるプログラム作成過程構築に客観性をもたせるためにも重要な情報である．発話プロトコルの一部を図 3.2 に示す．図中の「順番号：」に続く発話は観察者のもので，「xx 順番号：」に続く発話は被験者のものである．

```
    <ここから、課題７について>
  48:だいたい大まかなことが終ったから、今日よく聞きたいのは最後の、記憶に
    あるだろうから、最後の課題です。一応、
fm48:これもgoto文を使って終ったんですよ。
  49:そうですね最後ね。これがモニタ報告書ですから、思い出してください。
    この課題はこれですね。
    で、課題をもらって考えてプログラムを作成して行きますよね。それをね、
    どういう風に考えていったかということをね、話して欲しいんです。
    私情を交えてもかまいませんがね。できるだけ詳しく教えて欲しいんですよね。
fm49:これはたぶん、これをまず見て（課題）これを機械にまず分かってもらえば、
    映画の題名、その間だけ足し込めば出ますよね、それぞれ同じもの同士を、
    これはawkでやってて、さっきみたいにどんどん足し込めばできるんですよね、
    だからこれを機械に認識してもらえればできるかなっと思ったんですよ。
  50:映画名をね。
fm50:そうです。その区別ができれば。だから意外とすぐできるような気がしたん
    ですよ。それに最後の合計もこれを全部縦に足せばできますよね。だから配列
    を使って考えたらできるなっと思って、実はこの中身（作成の手引）を見なく
    てもできるんじゃないかなと思って、結局どこを読んだのかな、あーこれです
    よね、同じ間は繰り返せがあって、ファイルをオープンしたりクローズしたり
    するのも慣れていないけれど、オープンしてクローズさえできればいいかなっ
    と思ってやり始めたと思います。
  51:オープンして、、なんて？
```

図 3.2　発話プロトコルの実例（一部）

（4）　プログラム生成過程調書

上記 3 種類のデータを基に，学習者が課題を読んだ直後からプログラム作成完了までの過程を，観察者が再構築し文書に記述したものである．これは学習者を中心にした作成過程の大きな流れを掴むと同時に，プログラム生成過程のモデル化の際には，モデルの検証データとして使用できる．プログラム生成過程調書の一部を図 3.3 に示す．

```
被験者：
日　時　：1993.12.21 10:30～
課　題　：映画別入場者数集計表作成


１、課題を読んで、出す値は何かを見ていった。
  この課題では、映画別の合計人数と横の合計と総合計を出せばよい。
  そのうち、映画別の合計人数さえだせばあとはそれから出すことができるから、
  この課題は映画別の合計人数をだすことがすべてであると思った。

２、次に処理をどのように分けてするかを考えた。
  「計算」と「印刷」を分けることがすぐに頭に浮かんだ。つまりファイルから読ん
  だデータから映画別の人数をすべて集計して結果を出しておいて、それを後で印刷
  するという方法である。
  この方法だと後で見てここが何をしているところかが良く分かるので、できるだけ
  計算した結果を配列にいれる処理とそれを利用する処理に分けて考えるようにして
  いる。この方法でうまくいくプログラム課題は多いし、それにこの方法だと、計算
  した結果が後でいろいろに利用できるからいいかなっと思っている。
```

図 3.3　プログラム生成過程調書の実例（一部）

3.3.2　観察方法

プログラム作成時の観察方法として従来から行われている方法は，thinking-aloud（思考口述）[43),44),45)]である．この方法は，プログラムを作成しながら学習者に発話させ，それを記録するという方法である．それに対して，本研究では，プログラムを作成した後で学習者に自分の作成したプログラムを見せながらそれについて発話するという方法で行った．発話プロトコル収集の注意事項として「遡及言語報告には注意せよ」[43)]というものがあり，行動後の発話によるデータ収集は一般的には行われない方法である．しかし，プログラミングという作業は高度な論理的な思考が中心の作業であり，そのような作業をしながらの発話は学習者に多大な負担を課すことになり長時間におよぶプログラム作成過程を自然なかたちで維持できないと考え，作成後のインタビューという方法を採った．後日内観を要求するという方法は，人間の記憶に頼るという弱点がある．記憶に残っていることでも，時間的な経過が前後するとか，何を契機にそれを想起したかなどが不明瞭になるなどの可能性がある．これらの弱点を少なくし，thinking-aloud に近い形で内観させるために，本研究ではオンラインプロトコルと作業報告書を活用した．そのため，内観を発話するという意味では一種の thinking-aloud であるが，プログラム生成後に実施した点が異なる．このような発話プロトコル収集方法は，学習者のプログラム作成が自然の形で行えること，被験者の口述内容を観察者が推測し，それをすぐに被験者に確認できるという点で，プログラミング行動を観察する一手法として有効である．一方，この観察方法の欠点は，データ取得に非常に長時間を必要とすることである．

3.4　プログラム生成過程の分析

3.4.1　プログラム生成過程のモデル化のためのデータ

課題 17 のプログラム作成過程で得られたオンラインプロトコルと作業報告書を基に各被験者にインタビューした発話プロトコルを取得した．さらにそれらを統合してプログラム生成過程調書（図 3.3 参照）を作成した．課題 17 に関して各被験者から収集したデータを表 3.3 にまとめる．

表 3.3　プログラム生成過程分析のために収集したデータ（課題 17 に関して）

被験者	オンラインプロトコル（プロトコル数）	発話プロトコル（プロトコル数）	プログラム生成過程調書（段落数）
nt	31	108	25
yt	59	147	26
kt	7	273	18
sm	26	357	25
yk	46	466	21
fm	34	442	20

表中のオンラインプロトコル数は，学習者がプログラム作成過程で翻訳実行した回数に相当し，翻訳したプログラム数にも相当する．また，発話プロトコルは，インタビューにおいて，学習者が質問への回答や説明などで発話した回数に相当する．プログラム生成過程調書は，観察者が学習者のプログラム作成過程を記載した文書である．段落数は，その文書の段落の数に相当する．これらのデータからみると，発話プロトコル数が演習でプログラム作成した被験者と観察実験としてプログラム作成した被験者とでは,大きく異なる．演習課題で作成した被験者は，相対的に発話に対し慎重であった．データ数が少ないので一概には言えないが，評価されるプログラム作成に対する発話は，インタビュアーが評価者である場合，被験者に特別な緊張感を与えるのかもしれない．今後上記のような方法でデータ取得をする場合は，これを念頭に置き観察する必要がある．

上記のデータを基本にして，学習者のプログラム生成過程モデルを構築する．以下でこれらのデータを参照する場合，発話プロトコルは，sm11 のように小文字の被験者名とプロトコル番号で示し，プログラム生成過程調書は SM11 のように大文字の被験者名と段落数で示す． オンラインプロトコルは，Psm11 のように小文字の被験者名の前に大文字の P を付け，被験者名の後にプログラム版番号を示す．

3.4.2　ゴールとプラン

本研究では，学習者のプログラム作成過程の表現を Spohrer らの用いたゴール／プランモデルの枠組みで捕らえている [27]．この方法では，作成しなければならないプログラム機能をゴールと呼び，それを実現する方法をプランと称する．ひとつのゴールに対していくつかのアルゴリズムが存在するため，課題のゴールは複数のサブゴールに分解される．したがって，プログラムはひとつの課題に対して抽象的なトップゴールから具体的なアルゴリズムに至るまでの階層的なひとつの木の構造をしていると捉える．ひとつのゴールに対

して複数のプランが存在する．したがってプランもゴールと同様な抽象的なプランからプログラムコードに対応するプランまでを階層的な木構造をしていると考える．プランは，プログラムとして正しいプランだけでなく，プログラム作成者が行動したすべてを何らかのゴールのためのプランと捉える．ゴールとそれに対応するプランを，ひとつの木で表現したものをゴール／プラン木と呼び，学習者のプログラム作成過程の行動をこの木で表現しようとするものである．

本研究に先立ち，このゴール／プラン木を用いて学習者のプログラムの誤り分析などを実施した．この方法は，学習者の誤りの箇所が偏在していることを示すのに非常に有効な方法であった．ゴールとプランの実例として，学習者が作成したプログラムの誤り分析で作成したゴール／プラン木の例を図 3.4 に示す．図 3.4 の「G:始め」のように G:で始まるのはゴール名であり，「P：3 重ループ」のように P：で始まる名前はプラン名である．

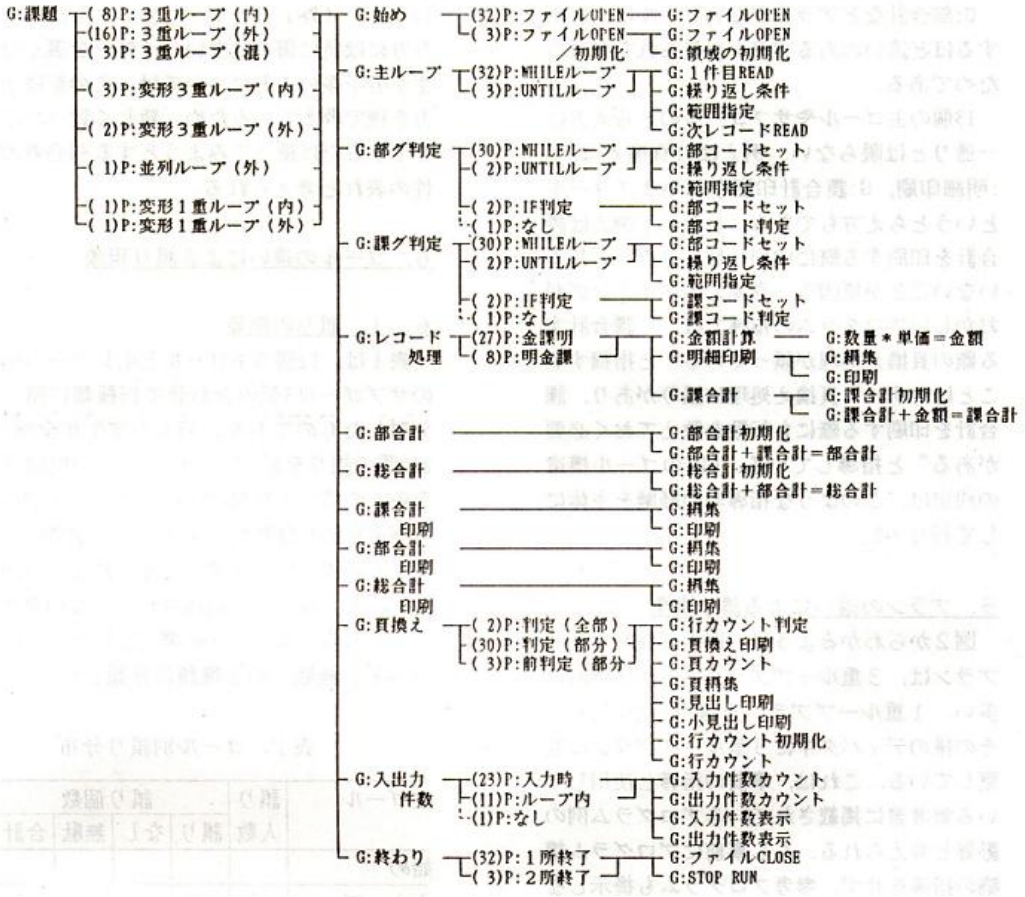


図 3.4　ゴール／プラン木によるプログラム生成過程の表現

学習者はプログラムを作成する過程で，上記ゴールとプランを次々に想起し，プログラム作成が完了する段階で，ひとつの木が形成される．そのため，次節のプログラム生成過程モデルでは，全面的にゴールとプランを用いて説明する．

3.4.3　プログラム生成過程モデル

3.4.3.1 基本モデル

プログラム生成時の学習者の行動は，プログラム課題を与えられた段階から捉えると，準備，ゴール計画，プラン生成，実行，評価の５つのフェーズで構成される．これらの関係を図 3.5 に示す．図中のゴール計画，プラン生成，実行，評価は，生成予定のゴールがなくなるまで繰り返される．この一連の繰り返しをサイクルと呼ぶと，このサイクルをどの媒体を使用して実施するかにより次の 3 種類が考えられる．

（１） サイクル１

特定のゴールを実現するために，そのプランを考案し，エディタを使ってソースプログラムを書き換えてコンピュータ上で翻訳実行し，その結果を評価する．

（２） サイクル２

特定のゴールを実現するために，そのプランを考案し，エディタでソースプログラムを書き換えるが，コンピュータ上では実行させず，自分の頭で仮想実行を行う．そしてその結果を評価する．

（３） サイクル３

特定のゴールを実現するために，そのプランを考案するが，ソースプログラムは書き換えず，書き換えたものとして自分の頭で仮想実行をする．その結果を評価する．

これら種類の異なるサイクルが，プログラミングの全過程を通じて何重にも繰り返されるのがプログラム生成過程である．一般的には，プログラム生成の初期段階でサイクル２とサイクル３を多く使用し，デバッグ段階に入るとサイクル 1 が多用されることになる．どのサイクルを多く使用するかは，学習者によっても異なる．例えば，オンラインプロトコルが極端に少ない被験者 kt は，「…，プログラムが出来たとして，こう仮のデータを入れて実行してみるっていうか，…，まー動く順序を自分がコンピュータになったつもりで考えてみるっていうか…」（kt45：被験者 kt の発話プロトコル 45 番を意味する），「あまり，何か，エラーがばーって出るのが好きじゃないっていうか，…」（kt184）と述べてお

り, サイクル 1 は少ないがサイクル 3 が多いことが推測できる. それに対して, fm は「…, 全部直して思い切ってできるという自信があったらやるけど, …でちょびちょび直すしかないんです」(fm363),「…機械に怒ってもらうほうがいいね. ちょっと直して怒ってもらって」(fm364) と述べており, どちらかというとサイクル 1 を多用していることが伺える.

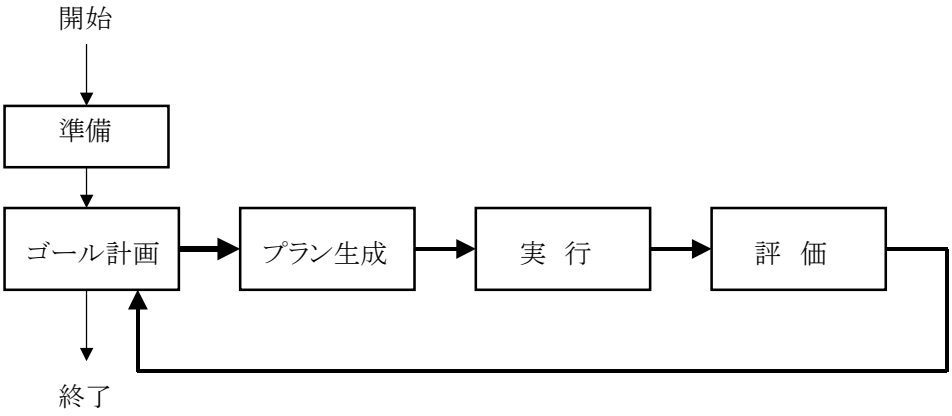


図 3.5　プログラム生成過程の基本モデル

3.4.3.2 準備フェーズ

準備フェーズでは, 学習者は与えられた課題のプログラムを作成するために必要な知識を予想し, ゴールに対しての難易度を見積もるなどして見通しをつけようとする. 例えば,「映画別の人数さえだせば, …それを出すことがこの課題のすべてだ」(SM1:被験者 sm の調書の段落 1 を意味する),「…そんなに難しくないなと思った. 処理の場合分けがすべてで, あとは計算だけだから…」(KT1),「この課題は今までとは違うから, …, もう少し情報を集めてからやろう」(yt2) などがこの状況を示している. 学習者は, 自分の経験的知識を参考にしながら課題ゴールを数え上げ, 各ゴールの重要性や難易度を予測し, それに必要な知識を準備しておこうとする. 準備フェーズのプロセスは, 次のようになる.

（1） 課題文から想定出力を形成する.

（2） 想定出力から, 必要な課題ゴールを抽出し, ゴール予定表に格納する.

（3） 抽出した課題ゴールにゴール特性を付加し, 大まかな方針を決める.

（4） 課題ゴールに必要とされるプラン知識を準備する.

学習者が課題から抽出する課題ゴールは, 完全なものではなく, 時には欠落するゴールもありうる. 例えば, 被験者 yk は総合計を計算し印刷するゴールを最後まで忘れていた.

これは上記（1）で正確な出力結果のイメージが形成されなかったことを意味している．また学習者は，この段階で課題の中の重要なゴールはどれか，難易度の高いゴールはどれか，既知ゴールか否かなどを判断しようとしている（SM1，KT1，YK1，FM2，NT2，YT1）．これらが，課題ゴールのゴール特性となる．これらのゴール特性がプログラム作成方針に影響を与える．方針としては，難易度の高いゴールから実現しようとする方針と，被験者 nt のように容易に実現できるゴールから実現しようとする 2 通りがある．しかし初心者の場合，難易度の予測を誤り「簡単にできると思った．これは課題 12 と似ているから…」（NT2）と開始したが，その後すぐに行き詰まり状態になる場合も多い．ここで準備される知識は，それ以前に学習者が作成したプログラムで活用したプログラミング知識が想起されることが多い．

3.4.3.3 ゴール計画フェーズ

ゴール計画フェーズでは，ゴール予定表をその直前の評価から更新し，ゴール予定表を順序化する．その結果，最も優先度の高いゴールがひとつ選択され，次のプラン生成フェーズに渡される．このゴールをターゲットゴールと呼ぶ．ゴール予定表にゴールがなくなり，ターゲットゴールが選択できなくなるとプログラム生成を終了する．ゴール計画フェーズのプロセスは，次のようになる．

（1） ゴール予定表のゴールを更新する．

（2） ゴール予定表のゴールを順序化する．

（3） 優先順位の最も高いゴールを選択する．

直前の評価が未完であっても，繰り返し同じゴールが選択されることは多い．特に，評価結果が不満足であっても出力結果が変化するプランが生成できる間は，そのゴールを中止して別のゴールに移ることは少ない．同一ゴールを連続して選択しても，4，5回くらいで成功すれば，学習者は「すぐに出来た！」という感想を持つ（fm127，yk7）．これ以上になると少しずつ行き詰まりと感じ始め，原因を調査しようとしたり，別のゴールを試みようとする．

ゴールの順序化は，ゴールの重要度，難易度，緊急度に左右され，通常は準備フェーズで決められた方針に従って決定される．しかし，途中で追加された課題に依存しないゴール（後述する戦術に関するゴール）は，優先的に選択される．困難なゴールから実現する方針を採っていた被験者 fm は，「終わるのが一番（実行が正常に終了するという意味），

計算するのは 2 番（計算結果が正しいという意味），3 番目がきれいにすること（出力形式を整える）.」（fm335）の明確な指針を決めていた．このようなゴール選択のアルゴリズムは学習者により複数存在し，この違いが同じ課題のプログラムを作成する場合でも異なったプログラム生成過程を辿ることになる．

3.4.3.4 プラン生成フェーズ

プラン生成フェーズでは，ターゲットゴールをキーとしてプランを想起する．ここで想起されたプランとひとつ前のサイクルまでに生成されていた現プログラムとをマッチングして, 新しいプログラムを生成する. プラン生成フェーズのプロセスは, 次のようになる.

（1） ゴール名をキーワードにしてプランを想起する.

（2） 想起されたプランと現在のプログラムをマッチングして, 新プログラムを生成する.

（3） 新プログラムを現在のプログラムと置き換える.

ここで想起されるプランは，プログラムコードとしてのプランから抽象的なプランまで非常に幅が広い．プログラム風に記述されているがプログラムコードではないプランもいくつか観察されており，学習者はゴールに対応する正確なコードがすぐに思い出せない場合や, ターゲットゴールを後回しにしようとする場合などに, 仮のプランを生成していた. 複数のプランが想起された場合，既知プランから先に選択されることが多い．学習者にとって，それが最も成功する可能性が高いためである．既知プランがない，あってもすでに失敗している時には，未知のプランを試みる．未知のプランであっても，教師，友人，参考書などですでに確認してある場合には，それが優先的に使用されることもある．

その他，プログラムのインデントを変更するとか，出力の文字表現を変更するなど，そのターゲットゴールに直接関係のないプランが生成されることがある. それは,「もう悪いとこが分からなくなると，どっこも打てないでしょう．で，chirdren を kids に変えたと言うことです.」（fm251）のように有効と思えるプランが全く想起できなくなると，プログラムに大きな影響を与えないようなプランを生成しようとしていた.

3.4.3.5 実行フェーズ

プラン生成フェーズで作成されたプログラムは，実行フェーズで実行される．すでに述べたようにプログラムは，コンピュータ上で実行される場合と作成者の頭や時には教師や友人の頭で実行される場合がある．いずれかの方法で実行された結果が記録される．実行

フェーズのプロセスは，以下である．

（1） プログラム実行媒体を決定する．

（2） 新プログラムを実行する．

（3） 実行結果を記録する．

課題作成の初期段階では，プログラムの生成がある段階に達するまでコンピュータ上でプログラムを実行しないことが多い．また，学習者が頻繁に使用する確信度の高いプランなどもコンピュータ上で実行しないことがある．自分の生成したプランを教師や友人に正しいことを確認させるという方法で実行する場合もある．このように学習者は多様な媒体を使って，自分のプランを実行する．いずれの媒体で実行された場合も，実行結果として実行状態や出力結果，連続実行回数などが記録される．

3.4.3.6 評価フェーズ

実行フェーズで記録された結果からプランが評価される．この評価結果が，次のサイクルのゴール計画フェーズに渡される．評価フェーズのプロセスを次に示す．

（1） 一つ前の結果と今回の結果の差をとる．

（2） 現プログラムを評価する．

（3） 実行フェーズの記録に評価結果を付加して保存する．

評価フェーズに至るまでのサイクルで保存される情報としては，実行状態，連続実行回数，出力結果差，ターゲットゴール，ターゲットプラン，これまでのサイクルでの評価，友人や教師の評価基準などがある．これらを基にして学習者は実行結果を評価する．学習者は今回の実行結果だけでなく，「結局さっきの終わんないやつに戻ったんですよ」（yk384），「そうですよ．ひっくり返す前のに，元に戻ったんですよ，…」（yk385）などから明らかなように，それまでの経過での結果とも付き合せて評価する．

3.5 学習者のプログラム生成過程の特徴

3.5.1 プログラム生成過程で用いられるプラン

ゴールは，課題が決まると一定の範囲で自動的に決定できる．しかし，ひとつのゴールに対して想起されたプランの違いによってサブゴールは異なる．したがって，ゴールもまたプログラム作成開始段階ですべてが決定されているわけではない．それに対してプランは，学習者のプラン知識から誤りプランも含めて作り出されるものであり，実に多様なも

のが含まれる．これらを次の 5 種類に分類した．

（1）　参照プラン

テキストに掲載されている，あるいは教師や友人などから教えられたプランである．初めて教えられた命令やアルゴリズムで，学習者がまだ一度も使用したことのないプランが参照プランであり，教えられたままのプランを使用することが多い．

（2）　実例プラン

学習者が一度使用したことのあるプランであるが，まだ完全には習得していないプランである．以前作成したことのある課題の一部などがこのプランに相当し，使用したことのあるプランをテンプレートにして新たなプランを作り出すことが多い．

（3）　既知プラン

既に良く知っており，一般化されたプランである．プログラミング学習の初期に学習した学習者が良く使い慣れているプランなどが相当する．既知プランが徐々に蓄積されることがプログラミングの学習の前進とも言える．

（4）　組立てプラン

自分が知っている命令を組み合わせて作成したプランである．課題文や教師の説明などをもとに，知っている命令や既知プランを組み合わせて新しく作成したプランである．プログラミング教育では，組立てプランを学習者が作成することを期待している．

（5）　コードプラン

使用するプログラミング言語の命令に対応した最も基本となるプランである．変数の宣言，代入文，制御命令，算術式など，学習者が知っている言語の基本的な命令が相当する．これらは，プログラミング教育の初期段階で教授されることが多いが，それぞれの命令に許される多様な記述の違いをすべて指導することが必ずしもできず，学習者の言語学習に依存する部分も多い．コードプラン以外のプランには，抽象的なプランも含まれるが，このコードプランはプログラミング言語に強く依存したプランであり，他の 4 つのプランを構成し理解するために用いられる基本的なプラン知識となる．

3.5.2　プログラム生成のための戦術プラン

3.5.2.1 戦術プランの種類

学習者のプログラム生成過程を詳細に観察すると，学習者の行動は前節で示したプランに当てはまらない行動が数多く見られた．前節までに述べたプランは，あくまでも課題プ

ログラムの一部を構成するプランである．ところが，最終的なプログラムには不要であるようなプランをプログラム生成過程途中に使用する現象が多く観察された．それらのほとんどがデバッグ時に行き詰まり打開の策として使用されており，そのためそのプランはプログラム生成に多大な影響を与えている．そこで本研究では，前節までに述べたプログラム課題のゴールに対応したプランを課題プランと呼び，それ以外で学習者がとる行動のすべてを戦術プランと称してその詳細を分析した．その結果，以下の 7 種類の戦術プランを抽出した．

（1） 仮プラン策

特定のゴールには,あえて正確でない,たとえば覚書のようなプランを生成する策である．

（2） 後回し策

特定のゴールに対するプランを意識的に生成しないで後回しにする策である．

（3） ゴール変更策

行き詰まり状態にあるゴールを一時的に中断し，実現可能なゴールへターゲットを変更する．

（4） 調査確認策

プログラムの実行推移が推測あるいは確認できるプランを挿入する．

（5） 奇襲策

どのような変化が起こるかを試みる実験的なプランを生成する．

（6） 安全無策

結果には何も影響しないようなプランをあえて生成する．

（7） 復帰策

現在の行き詰まり状態のプランよりはマシであった以前のプランへ戻す．

学習者は，課題プログラムを完成するまでに課題プランの使用だけでなく，上記のような戦術プランを使用しながらプログラムの生成を進めている．

3.5.2.2 プログラム生成過程における戦術プランの使用

被験者 6 名のプログラム生成過程において，使用されたプランを種類別にプロットしたのが表 3.4 である．表 3.4 では，プログラム実行結果の状態を 7 段階に分けて表示している．このレベルは，「L0：翻訳エラーがある」から「L6：正しい結果を出力」までのプログラムの状態であり，図 3.6 に示すような決定木で決められている．

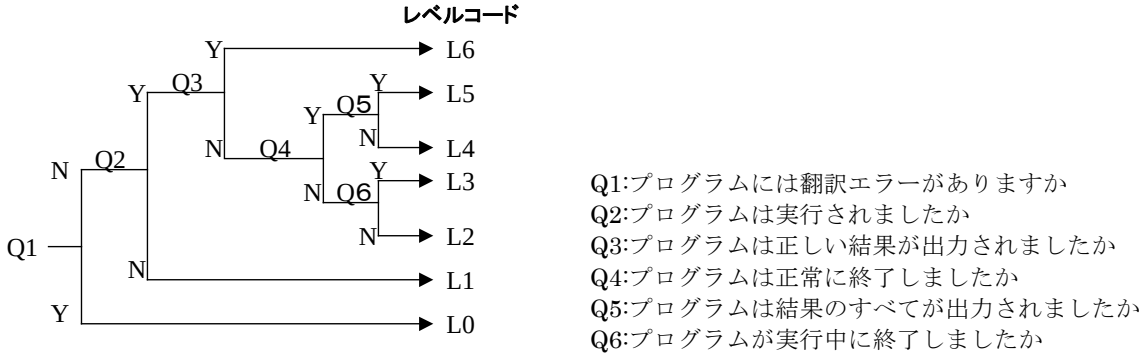


図 3.6　実行結果レベル判定の決定木

表 3.4 の横の数値は，学習者のプログラム生成順序を示しており，学習者がプログラム完成までに翻訳実行した回数でもある．そのため学習者によって，最も長いもので 50 回を超えており，少ない学習者は 4 回で完成させている．表中の●は，通常のプログラムプランを生成しており，◎はプログラムが完成していることを示している．◇の印のある箇所は，学習者が戦術プランを使用している箇所である．

3.5.2.3 課題ゴール別の戦術プランの使用

学習者は，課題プログラムのどのような箇所で戦術プランを使用しているかを調査したものが，表 3.5 である．プログラムの使用箇所を課題プログラムの 18 個のゴールとコメントなどの合わせて 19 箇所で示す．課題ゴールとは，その課題を完成させるために必要な機能部分である．調査対象とした課題「映画別入場者数集計表」のゴールは，宣言，始め処理，見出し印刷，主ループ，グループ判定，入場者数計算，入場者数印刷，総入場者数計算，総入場者数印刷，終わり処理の 10 個が必要である．それらのうち，始め処理，主ループ，グループ判定，入場者数計算，総入場者数計算のゴールはサブゴールに分解している．

表 3.4　被験者別プログラム生成過程での戦術プランの使用

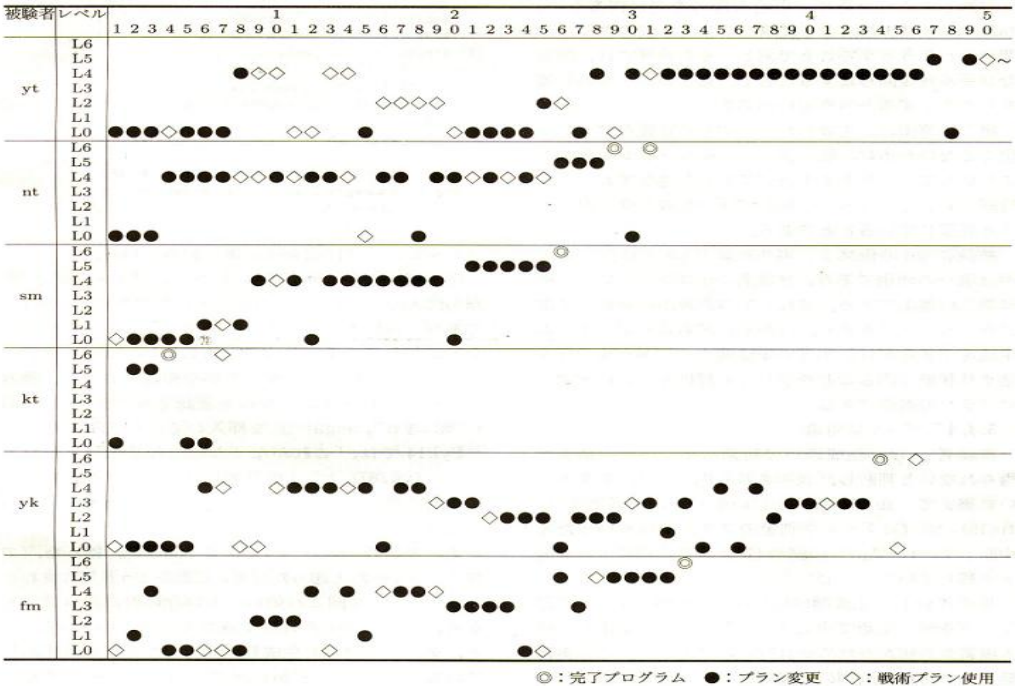


表 3.5　課題ゴール別戦術プラン使用傾向

ゴール名	仮プラン策	後回し策	調査確認策	奇襲策	安全無策	復帰策	合計
宣言							
始め／ファイルオープン					yt(1)		1
始め／初期化							0
見出し印刷					fm(1)	kt(1)	2
主ループ／レコード入力			yt(5),yk(6)		nt(3)		14
主ループ／繰り返し条件		yk(1)		yk(1)			2
主ループ／繰り返し範囲		yk(1)	yk(1)	yk(1)			3
グループ判定／映画名代入			yt(2),yk(1),sm(2)			fm(1)	8
グループ判定／映画名比較	fm(1)		yk(7)	yk(1)		yt(1),fm(4)	14
グループ判定／グループ範囲			yt(2)	yk(1)	yt(1),nt(1)		5
入場者数計算／種別計算			yt(9),yk(1)				10
入場者数計算／1 日計算			yt(3)		nt(1)		4
入場者数計算／初期化			yt(3)				3
入場者数印刷			yt(4)		yt(2),nt(1)	kt(1)	8
総入場者数計算／計算							0
総入場者数計算／初期化							0
総入場者数印刷	sm(1)					kt(1)	2
終り／ファイルクローズ							0
コメントなど					yt(4),nt(2),fm(1)		7
合計	2	2	46	6	18	9	83

3.5.3　戦術プランの内容と意図

前節までに示したように，学習者はプログラム作成中の，特に行き詰まり状態で戦術プランを多く使用している．使用されたそれぞれの戦術プランには，学習者の意図がある．学習者に共通している意図は，プログラムの生成を進めるため，あるいは少なくともプログラム生成を後退させないために使用していることである．以下にそれぞれの策の意図をその実例とともに示す.

3.5.3.1 仮プラン策

仮プラン策は，ゴールに対する仮のプランを意図的に生成しておき，後で正確なプランに置き換えるという戦術である.

被験者 sm は，生成計画を立てる段階で，映画別入場者集計のゴールさえできれば，それを印刷するゴールは簡単であると判断していた.「総合計とかは後でいいなと思っていたから」（sm82），「ここらで確認しておこう」（sm84）として，コーディング段階の G:総入場者数印刷には仮のプランを記述している（Psm1）.

被験者 fm は，ほとんどのゴールをプログラム生成計画通りに記述しているが，G:グループ判定／映画名比較は正しいプランが生成できず,「こんなコマンドってないと思うんですけど…，とりあえずやっていって…」（fm74）と，次のようなプランを記述している（Pfm1）.

While (title[0]=same){……}

被験者 fm はこのプランが正しいとは思っておらず，最初の翻訳で「絶対ここが」（fm97）エラーになることを予想して，デバッグの初期段階で訂正している．このように，学習者はプラン生成が困難な場合，あるいは容易なものであっても，そのゴール以外のプログラム生成を先に進めたい場合,仮のプランを作成し,後で正しいプランを生成しようとする．仮プランは生成を忘れないための覚書になり，プログラム生成を停止することなく学習者に正しいプランを考える時間を与えてくれる.

3.5.3.2 後回し策

後回し策は,学習者のプラン生成計画には入っているが,仮のプランを置くこともせず,意図的に後回しにする策で，仮プラン策の特例とも考えられる.

被験者 yk は,「とりあえず，映画が変わったらっていうのができているかって，映画が

変わったら……というのを見ようと思って」（yk210），最初 1 つの映画グループの処理だけを実現するために，G：主ループを記述せずに G：グループ判定のプランが完成したあとで，G：主ループのプランを記述しようとしていた．その理由は「この課題は映画の名前が変わったらっていうとこが大きなポイントと思っているんですね．それで（最初に）区切れるか」（yk255）ということを最優先させるために，G：主ループを後回しにした．

仮プラン策や後回し策の意図として，大きくは 2 つ考えられる．第一は，「肝心な部分を先に作ろう…」（sm82）という意図である．その背景には，このゴールは後回しにしても簡単にできるという判断がある．しかも，対象とするゴールをとりあえず少なくすることにより，重要なゴールや困難なゴールに専念することができる．第二は，正確なプランが想起できないために，とりあえず仮のプランを記述しておいて，他のゴールのプラン生成を先に進めようとする意図である．これにより，学習者自身が調査しないといけない箇所を明確にしたうえでプログラム作成を先に進めることができる．被験者 sm の仮プラン策や被験者 yk の後回し策は第一の理由であり，被験者 fm の仮プラン策は第二の理由の例である．

3.5.3.3 ゴール変更策

ゴール変更策は，現在取り組んでいるゴール（ターゲットゴール）が完成していないにも関わらず，他のゴールに取り組む行動を指す．

被験者 nt は，デバッグの初期段階でプログラムが正しい答を出力しないため，「表示される内容より，形をきれいに整えて，あとで中身が正しいかどうかを考えよう」（nt15）と集計するゴールが未完成であるのに，それを中断して集計結果を出力するゴールをターゲットゴールに変更している．

被験者 yt は，「始めて出力された集計表は，見出しと総入場者数の値がゼロの合計行だけであった．この実行結果……特に映画別の入場者数が全く表示されていないから，映画グループの判定がうまくいっていないのだと思ったが，その解決方法が浮かばない」（YT10）ので，G：グループ判定のサブゴールを次々に選択していくつかのプランを試している（Pyt9～）．

プログラム作成という問題解決は，解決すべきゴールが同時に複数存在することが特徴である．つまり解決すべき最終的なゴールは，必ずしもひとつではない．そのため学習者はどのゴールから実現をしていくかの戦略を立てる．しかし，それが行き詰まるとターゲ

ットゴールを変えることにより，何とかプログラム作成を続けようとする．学習者がゴール変更策をとるひとつの理由は，バグのあるゴールが不明な場合ゴール変更策をとり，どのゴールがバグの可能性があるかを探ろうとすることである．もうひとつの理由は，困難なゴールは一旦中断して容易なゴールを確実に生成することにより，バグのあるゴールに焦点をあてやすくすることである．いずれも行き詰まり状態を長期化させるのを避けようとするための戦術と考えられる．

3.5.3.4 調査確認策

調査確認策は，作成中のプログラムの動きを探るために採る行動をいう．

被験者 yk は，プログラム生成過程の前半で，出力される結果は正しいが一部の結果が出力されないという状態がかなり続いた．その間，3 回の調査確認策を試みている．例えば，G：グループ判定／映画名代入のプランが「これがちゃんとできているかなって調べ」（yk227）るために，図 3.7 に示すプログラムの(4)と(5)の命令を追加している（Pyk7）．その後，読み込み処理(1)の後に fscan()でデータが読み込まれているかどうかを確認するために同様の命令を追加している（Pyk10）．さらに，Pyk14 では「これがなくなったらどうなるんだろう…」（yk287）と，1 件目のレコードの映画名を比較する前の映画名代入プラン(3)を削除して実行を試みている．

```
        ……
infile = fopen(“eiga1993-12.dat”, “r”);        (1)
flag=fscanf(infile,”%d %d %d %s …);            (2)
strcpy(cinema,eiga);                           (3)
printf(“%s¥n”,cinema);                         (4)
printf(“%s¥n”,eiga);                           (5)
if(strcmp(eiga,cinema)=0);                     (6)
    {
        ……
```

図 3.7　調査確認策の例（Pyk7 の一部）

さらに，被験者 yk は，Pyk44 で「回った回った，（できて）良かったと思ったけど，これちょっと気に食わなかったから，2 回もこれ効いていもいないのに置いているから，括弧の中をはずしてみたんですよね」（yk397）と，一応完成した後で自分が以前に作成したプランを再度試みたりしている．しかし，「やっぱりエラーが出たんですよね．で，もと通

り入れて落ち着いたんですよ」（yk398）と納得し，プログラムの生成を終了している．

このような戦術をとる学習者の意図は，第一に自分で生成したプログラムの内部構造を理解しようとすることである．第二は自分の生成したプログラムの正しさを確認しようとすることである．これらによりバグのあるゴールを絞り，その打開策の手掛かりを得ることが目的である．

3.5.3.5 奇襲策

非常に大きな行き詰まり状態に陥った際に，その状態を変化させるために論理的にも飛躍する大胆なプランを実行してみる行動を言う．

被験者 yk は，Pyk20 でプログラムがほぼ完成したと確信をもったが，計算結果が一部正しくないため，調査確認策を使って計算結果の途中経過をトレースするプランを挿入した（Pyu22）．その結果プログラムが正常終了していないことを知った．いくつかのプランを試みたが，プログラムの異常終了を脱出できず， G：主ループと G：グループ判定のプランを交換するという，論理的に飛躍のあるプランを突然生成している（Pyk30）．「ここまで来るともうやけのような気がする」（yk335）と述べているように，長い行き詰まり状態から脱出するために，一か八かで起死回生を狙って奇襲の策を講じたと考えられる．

奇襲策の意図としては，第一に八方ふさがりの状態を一時的にでも変化させ，行き詰まり打開の機会を狙おうすることである．第二に思いつきプランを実験的に試みることにより，新しい展開を期待することである．結果的には良い戦術とは言えない場合が多いが，打開の手掛かりを得たいという学習者の意図が窺がえる．

3.5.3.6 安全無策

長い行き詰まり状態で打開策が全く見つからない時に，プログラムの実行結果に影響を及ぼさない変更を加える．これを安全無策とした．

被験者 fm は，G：見出し印刷のプランの表示する見出しの単語を，children から kids に変更している（Pfm19）．これに対して「もう悪いとこがわからなくなると，どっこも打てないでしょう．で…」（fm251）「大した意味ない」（fm247）ことをすると答えている．その他の行動としては，インデントを変更する，コメントを挿入したり削除したりするなどが観察された．

この策により，学習者も行き詰まり状態から脱出できるとは考えていない．だからとい

って不確実なプラン変更をして今以上プログラム生成を後退させたくない．しかし，次の手が見つからない．そこで安全なプランを実行させながら考える時間を稼ぐというのがこの戦術の意図である．

3.5.3.7 復帰策

復帰策は，行き詰まり状態でプログラムの修復が不可能な状態に陥った場合でも，今よりはマシな以前のプランに戻すことで最悪の状態から脱出しようとする戦術である．学習者は，奇襲策などあまり確信のないプランに変更するときに，その時点のプログラムを別に保存しておいたり，抹消する命令をコメントにして残しておくことをする．これは，その後のプラン変更に失敗した場合でも，少なくともここまでは戻れる安全確保をするためと考えられる．

被験者 kt は，非常に慎重に生成を進め，Pkt4 で既に課題が完成した．その後で，G：入場者数印刷をプリンターに表示するプランに変更しようと試みるが翻訳エラーとなり断念し，直前のプランに戻して完成とした．

この策の意図は，一定の段階まで達したプログラム生成を後退させないことにある．

3.5.4　戦術プラン使用上の特徴

前節までに示した 7 つの戦術プランをより明確にするために，各戦術間の共通点や特徴をまとめる．

(1) ゴール変更策と安全無策は，行き詰まり状態にあるゴールから焦点を外すという点は共通している．しかし，ゴール変更策は，行き詰まり状態が比較的が低い早期に使われる策であるのに対して，安全無策は全く打つ手のない最大の行き詰まり状態で使われることが多い．

(2) 調査確認策と奇襲策は，行き詰まり状態からの脱出を積極的に試みようとしている点は共通している．しかし，調査確認策はかなり冷静な試みであるのに対して，奇襲策はかなり追い詰められた状態で起死回生を図る策である．

(3) 学習者が多くの戦術プランを使用するのは，G：グループ判定のゴールで，このゴールはこの課題で始めて試みるゴールであった．学習者が未知の処理を試みる場合，それを確実に実現するために多くの戦術プランが使用されることが読み取れる．これに

よって新しいプラン知識が獲得される．

（4）被験者によって使用する戦術に違いが見られた．例えば，被験者 fm と nt は，安全無策や復帰策など生成後退を避ける策が多く使用されていたのに対して，被験者 yk や yt は，調査確認策や奇襲策など生成を前進させようとする策を多く使用する傾向が見られた．

戦術プランのうち，仮プラン策，調査確認策，後回し策，復帰策などは，エキスパートプログラマも頻繁に使用する策である．その他の奇襲策，安全無策，ゴール変更策などは，必ずしも積極的な効果が得られる策とは言えず，エキスパートはあまり使用しない．しかし，これらの戦術プランは，その意図から，学習者の行き詰まり状態を推測する貴重な情報であり，プログラミング学習者に支援を考える際に，学習者の状態を判断する指標となりうる．以上の分析から戦術プランの使用は，プログラミング学習を学習者自身が進めようとする行為の表れであると考えられる．

4. デバッグ戦術の学習に関する分析

4.1 戦術学習の調査目的

プログラミング教育においてプログラム作成を実践する演習の重要性は自明であり，学習者は教授された知識だけでなくプログラミングに関する多くのことをこの演習で学習する．前章のプログラミング生成過程分析で，初期段階のプログラミング教育を受けた学習者がプログラムをどのように作成していくかを詳細に分析した結果，学習者は教授されたアルゴリズムやプログラミング言語の知識を駆使してプログラムを作成するだけでなく様々な戦術を使用することが判明した．しかも，戦術使用の目的が行き詰まり状態を打開するための方策であることも明らかになった．そのため，戦術の使用は学習者自身がプログラム作成を自立的に進めるための手段となっており，学習者のプログラミング学習に非常に大きな影響を与えることが推測できる．この結果から，戦術をプログラミングの初期段階の学習者に積極的に指導することがプログラミング学習を支援することに繋がるという確証を得た．本章では初期のプログラミング学習者が様々な戦術をどのように学習しているかを明らかにする．

4.2 戦術学習の調査方法

4.2.1 調査対象者

調査対象者は，宇部フロンティア大学短期大学部情報システム学科のプログラミングに関する科目「プログラミング」と「プログラミング演習」を 2005 年度に受講した 6 名の学生である．これらの学生は，講義および演習の 2 コマ（1 コマ 90 分）を週 1 回受講し，14 課題のプログラムを Java 言語で作成した．対象となった学生は，調査以前に表計算やワープロなどの応用システムを利用する教育は受けていたが，プログラミング教育を受けた経験は全くなく，調査した科目を受講し始めてプログラムを作成した．いわゆる最も初期のプログラミング学習者である．学習者が作成した課題内容を表 4.1 に示す．

4.2.2 調査対象の戦術プラン

一般に知られているデバッグ時の戦術にはトレースがある．トレースは，エキスパートの技術者があらゆる場面で使用するシステム構築技術のひとつである．したがって何らかの形でプログラミング教育において指導されている可能性が高い．しかし学習者は，3 章

で明らかなようにトレースだけでなく多様な戦術を使用している．これらの戦術のうち，以下の 4 種類を対象に使用状況を調査した．

- 仮プラン策
- 後回し策
- 調査確認策
- 復帰策

上記の 4 種類を選択した理由は，これらの戦術はエキスパートが積極的に利用する策であり，学習者も学習しておく必要のある戦術であるためである．調査対象にしなかった奇襲策や安全無策は，学習者が行き詰まり状態で何ら策が無い時にとる行動であり，エキスパートにはあまり見られない策である．したがって，これら 2 つの策の使用は，学習者が相当の行き詰まり状態であることを判断する手掛かりにはなるが，学習者にこれらを積極的に指導する必要はない．また，ゴール変更策は，プログラミング学習の最も初期段階で作成する課題には，選択するほどの多くのゴールが含まれることが少ないために，この戦術を使用する機会は少ないと考え調査対象から除いた．

表 4.1. プログラミング戦術学習の調査課題

課題番号	課題名	主な教育内容
1	イニシャル表示	プログラムの基本構造，作成手順
2	自己紹介	変数
3	犬の体重を表示	配列，for 命令
4	犬の平均体重	配列，for 命令，合計処理
5	四則演算	数値入力処理，計算
6	利子の複利計算（１）	数値入力処理，計算
7	台形の面積計算	入力例外処理，計算
8	利子の複利計算（２）	再入力処理，計算
9	閏年の判定	if 命令，条件式
１０	天気判定（１）	if 命令，条件式
１１	天気判定（２）	if 命令，条件式，for 命令
１２	九九の表作成	多重繰り返し処理
１３	最大値と最小値	while 命令
１４	データの並べ替え	アルゴリズム

4.2.3　調査の演習環境

本研究では，プログラミング戦術に関する学習支援システムを 2004 年から試作した．これらのシステムは実際の教育で使用し，その利用状況を分析し，次の支援システム改良

に反映させるという手順で第 3 版まで製作した．本調査は，第 2 版の DESUS を使用した演習環境で実施した．このシステムには，プログラム開発に必要な基本的な機能があり，支援機能には学習者が支援を求めた時点のプログラムをもとにトレースを指導する機能が備わっている．したがって，この調査は学習者が演習環境にある支援機能を使ってトレース学習ができる状況で行われた．そのためこの演習では，特別なトレース教育を教師がすることはしなかった．なお，第 3 版のデバッグ支援システム DESUS の詳細は，「6　トレースを支援するデバッグ支援システム」で述べる．

4.2.4　分析データ

6 名の被験者から取得したオンラインプロトコル（翻訳されたすべてのプログラム），支援システム利用ログデータ，支援システム利用アンケートの 3 種類を収集し，主としてオンラインプロトコルデータから戦術の使用調査を実施した．

オンラインプロトコルデータは，UNIX のソースコードのバージョン管理機能（RCS）を使用し，学習者がプログラムの翻訳実行をするたびに自動的にそのプログラムを保存した．保存したプログラム総数は 1830 本である．

支援システム利用ログデータは，学習者別にプログラム名，指導方法，日付が記録される．学習者が支援を求めた回数は 32 回で，課題数は 18 課題であった．そのうち支援に基づきトレースを実行したのは 7 課題であった．

学習者が課題レポートを提出するたびに，支援を受けたか否か，支援を受けた場合役立ったか否かのアンケートを実施した．6 人の被験者のうち，支援をうけて「役立った」と答えたのは 4 名であった．残りの 2 名は，利用ログに残されているにもかかわらず，支援を「使わなかった」と答えている．このアンケートは，戦術使用時の学習者の状況を確認するために使用した．

表 4.2 にオンラインプロトコルデータと支援システムログの両データをまとめたものを示す．表中の数値は，学習者が課題プログラム完成までに実行したプログラム数であり，背景が濃い灰色になっている箇所は，DESUS のトレース支援機能を被験者が使用した箇所である．さらに表 4.2 には，トレース支援に基づいて学習者がトレース実行した箇所を下線で示した．

表 4.2　課題別被験者別のプログラム実行状況

被験者	課題1	課題2	課題3	課題4	課題5	課題6	課題7	課題8	課題9	課題10	課題11	課題12	課題13	課題14
1	4	30	31	45	43	13	40	23	13	6	54	91	24	45
2	2	13	42	8	20	18	30	5	12	17	39	45	10	145
3	3	8	12	23	6	20	16	19	7	1	5	7	12	13
4	8	ー	21	15	21	27	16	10	14	13	22	9	4	20
5	17	3	23	13	10	33	25	12	15	17	10	9	20	29
6	4	26	2	13	26	19	50	14	10	11	3	13	6	177

表 4.3　プログラミング戦術の使用状況

戦術名	被験者	課題3	課題4	課題5	課題6	課題7	課題8	課題9	課題10	課題11	課題12	課題13	課題14	合計
調査確認策	1	▲								▲		▲	●	4
	2			▲										1
	3													0
	4													0
	5	▲					▲							2
	6												▲	1
仮プラン策	1													0
	2													0
	3													0
	4													0
	5													0
	6													0
後回し策	1	●				●	●			●	●		▲	6
	2				●					●	●		▲	4
	3		●		●		●						▲	4
	4		●		●								▲	3
	5		●	●			●			●			▲	5
	6						●	●					▲	3
復帰策	1					●						●	●	3
	2		●			●							●	3
	3													0
	4													0
	5													0
	6				●								●	2

4.3　戦術プラン使用の状況

調査は，最初の課題 1 と課題 2 を除いた 12 個の課題プログラムを対象に行った．調査方法は，学習者のプログラムを時系列に再実行させ，そのプログラムの実行状態とともに戦術プラン使用の有無を記録していった．これらの調査結果をまとめたのが表 4.3 である．表 4.3 に示す●印は，学習者が独自に戦術を使用した箇所である．それに対して，▲は，教師の個別指導あるいは支援システムの指導でその策を使用した箇所である．

表 4.3 の結果から，学習者の戦術プラン使用状況は戦術によって大きく異なる．以下にこの調査結果に基づき戦術使用に関する分析する．

4.4　戦術使用に関する分析

最も多く使用された戦術は後回し策であり，使用総数は 25 回で全員の被験者が使用していた．次に調査確認策と復帰策が同じ程度の使用であった．今回の観察では，仮プラン策は使用された形跡がなかった．これは，今回の調査が最も初期段階のプログラミング教育で行われたことと関連すると考えられる．以下では，仮プラン策を除いた 3 つの戦術の利用状況を詳細に分析する．

4.4.1　後回し策

後回し策は，ゴールを忘れないために覚書として何かを記述しておく仮プラン策と使用目的が似通っている．いずれも段階的にプログラムを作成しようとする策である．今回の調査では，仮プラン策は確認されていないが，それとは対照的に後回し策はすべての被験者が使用していた．同じような目的をもつ策であるのに使用状況が大きく異なるのは，今回の調査が最も初期のプログラミング課題であるため，仮プランを置くほどではなかったと考えられる．初期の課題は，多くのゴールを実現する必要がない．そのために仮の命令を書くことはせずに，単にプログラムの一部の作成を後回しにする後回し策を多用したと考えられる．

後回し策を使う被験者のプログラム作成過程を追跡すると，学習者の慎重な行動が窺がえる．確実に実行される範囲，例えば以前に作成したことのある手続きなどを使って，その部分を確実に完成させる．その後に，与えられた課題の手続きを少しづつ追加変更する方法で実現しようとする．その際に後回し策が使われている．

課題 14 は分類の課題である．この課題の説明段階で，教師は配列データを分類する 2

通りのアルゴリズム，泡立ち法と単純選択法を解説した．このプログラムは，①配列に数値データを読み込む,②配列の数値を分類する,③並べ替えられた配列の数値を表示する，の大きく 3 つの部分から構成され,解説したアルゴリズムは②の部分に関するものである．このプログラムで分類が正しく行われるためには，データが配列に正しく格納されていることが前提になる．したがって，この課題の解説で，教師は①，③を先に完成させ，その後で②を完成させた方が良いと，作成手順をアドバイスした．表 4.3 に示すように課題 14 で全員の被験者が後回し策を使用しているのは，この事前指導の影響がある．しかし，後回し策はそれ以前の課題で全員の被験者が一度は使用している策である．つまり課題 14 のプログラム作成以前に学習者がすでに後回し策を知っていたことが課題 14 での指導が徹底した原因と考えられる．

以上から後回し策は，慎重にプログラムを作成しようとする学習者によって，特別な指導をしなくても使用される策である．

4.4.2　復帰策

復帰策は，3 人の被験者が 8 箇所で使用しているのが観察された．復帰策の使用状況から 2 通りの目的が考えられる．ひとつは，被験者が未知のプランを試しに使って，その結果を確認して元のプログラムに戻すような使い方で，実験的な実行をしてみようとする場合に観察される．もうひとつは，プログラム作成が行き詰まり状態で打開策が全く見つからない時に，今よりも良い結果が得られていた前のプログラムに戻すような使い方で，プログラム作成を後退させないために利用している．今回の調査ではほとんどが後者の目的であった．この復帰策も学習者自らが使用を開始している．

4.4.3　調査確認策

使用された戦術のうち後回し策と復帰策は,学習者が何の指導も受けずに自発的に使用を開始していた．これに対して調査確認策（主としてトレース）は，4 名の被験者が 8 個の課題で使用していたが，そのうちの 1 箇所を除くすべてが支援システムの指示に基づいて行われたトレースであった．支援システムの支援を受けずに課題 14 でトレースを実行した被験者 1 は，少なくともそれ以前に 7 回のトレース指導を 5 個の課題で受けており，そのうち 3 個の課題で支援が「役立った」とアンケートに答えている．したがって，被験者 1 は課題 14 を作成する時には支援によりトレースを学習していたと考えられる．つま

り支援システムによる支援が課題 14 のトレース使用に影響を与えたと考える方が妥当である．以上から，調査確認策に関しては，学習者自らが何の指導も受けずにこの策を利用し始めることは確認できなかった．この点が後回し策や復帰策の戦術とは大きく異なる点である．

4.5　トレースの使用がプログラム作成に与える影響

被験者 6 名のうちトレースを使用してデバッグをしていた 4 名の被験者から，比較的初期の課題 3 でトレースを実行していた被験者 1 と被験者 5 のプログラム作成過程を分析し，トレース使用がプログラミング学習へ与える効果を検討する．

課題 3 の内容は，配列にあらかじめ代入された犬の体重を格納された順に表示するものである．学習者は，この課題で始めて配列と for 命令を使用した．これ以前の課題で，Java プログラムの基本構造と変数について学習し，それらの知識が必要なプログラム課題を 2 個作成している．

4.5.1　被験者 1 のプログラム作成過程

被験者 1 の課題 3 のプログラム作成過程を図 4.1 に示す．この図は，プログラムの実行結果の状態を翻訳エラー，実行時エラー，実行の 3 段階に分け，プログラムが完成するまでの推移を示したものである．プログラムの実行状態を〇で示し，出力結果が同じ場合は状態を遷移させず，単に線だけを記述している．したがって，線が多重になっている箇所は，プログラムの実行結果が変化していないことを示しており，この重なりが多い箇所は学習者にとって行き詰まり状態である可能性が高い．

被験者 1 が支援を受けたのは，図 4.1 の状態 I の最後であり，支援に基づいてトレースを実行した結果が状態 m である．状態 I は，8 回のプログラム修正実行を試みているが，プログラムは実行されるが出力結果が何も表示されないという状態である．この段階で被験者 1 は支援システムを利用し，「プログラムは何も出力しない」で「プログラムが最後まで実行されているか否かが分からない」と答えている．それに対して支援システムは，学習者のプログラムを用いて，プログラムが正しく終了しているか否かを確認するトレースを指導している．その指導に基づいて実行されたプログラムが図 4.2 のプログラムである．図 4.2 の←←に示す出力命令がトレースを実現する命令として支援に基づき学習者が挿入した命令である．

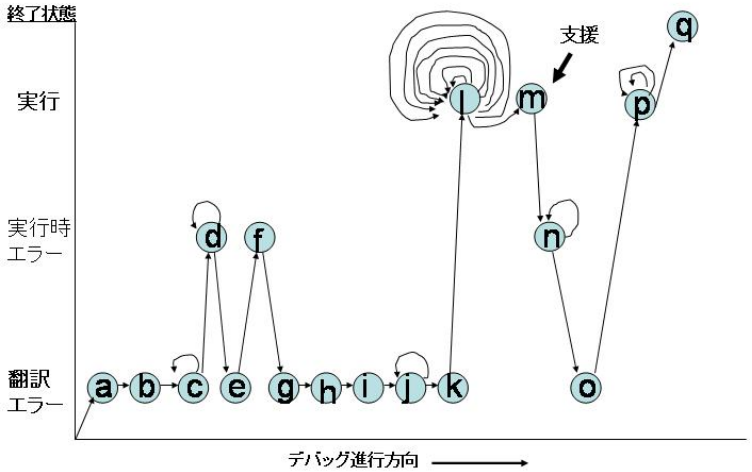


図 4.1　被験者 1 の課題 3 プログラム作成過程

```
public class DogTaijyu1{
        public static void main(String args[]){
        double taizyuu[]={12.5,12.0,15.7,14.6,13.5,12.9,11.3,10.8,8.0};
        int n;

        for(n=10;n<0;n--){
        System.out.println("inunotaizyuuha"+(int)taizyuu[n-1]+"desu");
        }
System.out.println("end of program");  ←←支援システムの指示による命令
}

}
```

図 4.2　被験者 1 の課題 3 プログラム例

このトレース結果から被験者 1 は,「プログラムの実行が行われているかどうかが分かった」,「私のプログラムに何かの問題があることが分かった」とアンケートに記述している．つまり被験者 1 は，このトレースでプログラムは最後まで実行されているが，結果が出力されていないことを始めて確認し，これが契機になり，繰り返し条件の比較演算子の誤りを発見しプログラムを完成させている．

被験者１は，トレースにより自分のプログラムで不明であった実行状態を知ることにより，次のデバッグ段階に進んでいる．その後５つの課題で７回も支援システムを利用し，３つの課題でトレースを実行している．これにより被験者１は，課題 14 では支援を受けずに自らがトレースを用いたデバッグをしていた．

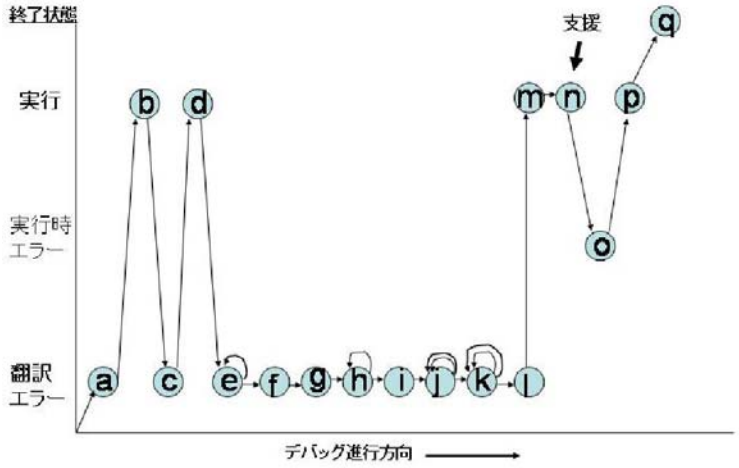


図 4.3　被験者 5 の課題 3 プログラム作成過程

4.5.2　被験者５のプログラム作成過程

被験者 5 が支援を受けたのは，図 4.3 の状態ｍである．それ以前は長く翻訳エラー状態が続き，それからやっと脱出した状態ｍでプログラムは何も結果を表示しなかった．そこで支援を受けて状態ｎでトレースを実行した．被験者 1 と同様にその時点のプログラムの状態を「プログラムは何も出力しない」,「プログラムが最後まで実行されているか否かが分からない」と答えている.支援システムは被験者１への支援と同様のトレースを指導し，被験者 5 はトレースを実行している．その結果に対して，被験者 5 は，「プログラムはちゃんと実行されていることが分かったけど，なぜそれが表示されないのかまでは分からなかった」とアンケートに答え，支援システムについては良い評価をしていない．しかし，この実行を契機にして繰り返し条件を 2 通り変更してプログラムを完成させている．

被験者 5 は，誤り箇所の指摘がなかったことから支援システムを評価せず，課題 8 でもう一度支援を受けてトレースを実行しているが，その後は支援を受けていない．その後の課題で学習者独自にトレースを実行することもしていない．

4.5.3　トレースとプログラミング学習

被験者 1 と被験者 5 は，支援システムへの評価は異なるが，プログラム作成過程から見ると，明らかにトレースの実行がプログラム作成を進展させている．しかも，重要な点は，結果的に被験者自らが誤りを発見し訂正しプログラムを完成させるに至っている点である．プログラミング学習は，問題解決に関する学習である．保持する様々な知識を活用し組み合わせて問題を解決するという行動がプログラミング学習の重要な側面である．トレースの使用は，この問題解決過程で学習者自らが行動する手掛かりを与え，行き詰まり状態を脱している．このことからトレースを学習し活用することができれば，学習者は自立的にデバッグを進める可能性があることを示している．

調査確認策以外の後回し策，仮プラン策，復帰策は，プログラム作成を後退させないで安全に効率よく進めるためのセーフティネットのような策である．しかし，調査確認策のひとつであるトレースは，それとは異なり今までのプログラムの動きを変える．これにより学習者はプログラムを見る視点を変えることができる．それがバグ発見に繋がり，学習者の新しいプログラム理解にも繋がる．トレースが他の戦術と大きく異なるのはこの点である．しかもこの戦術は，プログラミング教育課程で積極的に指導をしないと他の戦術のように学習者自らが習得することが少ない戦術である．

5. トレース支援システムの必要性

トレースは，ソフトウエアの開発のあらゆる場面で使用される非常に手軽で有効な戦術である．プログラミング学習者に対しては，この戦術がプログラミング学習を促進させる可能性がある．したがって，できればトレースをプログラミング学習の早い時期に習得させることが望ましい．このような考え方から，プログラミング教育現場では何らかの方法でトレースの指導が行われている．本研究では，トレース学習支援システムを導入する以前のプログラミング教育過程で，学習者がトレースをどのように学習し活用するかを調査した．その結果から，トレース学習支援システムの必要性について論及する．

5.1 支援システムを使用しない環境でのトレース学習調査

5.1.1 調査対象

調査対象は，2003 年度の宇部フロンティア大学短期大学部情報システム学科で科目「プログラミング」「プログラミング演習」を受講した 7 名の学習者である．２つの科目は，1 科目 1 コマ（90 分）を連続して週 1 回半期間開講された．教育に使用された言語は Java で，実習は UNIX 環境下で実施された．いづれの受講者も，この科目で始めてプログラミングを学習した者である．2003 年度には 16 個のプログラム課題が与えられた．

5.1.2 トレース指導方法

トレース教育に関しては，実習時に，学生が教師に支援を求めた際，その行き詰まり状態打開にトレースが最適であると判断すると，支援を求めた学習者に学習者のプログラムを用いてトレースを個別に指導した．と同時に，その時点で，つまり最初にトレースを個別指導した時点で，学習者全員にトレースを使ったプログラムの調査方法を口頭で解説した．

5.1.3 調査方法と調査結果

学習者が課題プログラム作成時に翻訳したプログラムを UNIX の RCS 機能を用いて自動的に保存した．保存されたプログラム総数は 992 本で，それらを課題毎に再実行することにより，トレース使用の有無とそれが教師により指導されたトレースか否かを調査した．

これらの調査結果をまとめたのが表 5.1 である．表中の数値は，学習者がその課題完成までに実行した翻訳回数で記録として残されたプログラム数に相当する．また，濃い灰色

の背景の箇所では，教師の指導によりトレースが実行された箇所である．表 5.1 から分かるように，課題 11 で学習者 f にトレースを指導し，この時点で全員の学習者にトレースという方法を解説した．教師がトレースを直接学習者に個別指導したのは，この 1 回だけであった．

教師がトレースを指導した以前にトレースを実行した学習者はおらず，トレース解説以後では，課題 16 で学習者 f だけが独自にトレースを実行していた．トレースは，プログラム作成の行き詰まり状態が使用する動機となり，行き詰まり状態がなければ使用しないことも考えられる．しかし，2003 年度の課題 16 は分類の課題で，すべての学習者が所定の期限内に完成に至っていなかった．したがって，どの学習者も行き詰まり状態であったと推察できる．それにも関わらず学習者 f 以外はトレースを実行した形跡がない．

表 5.1　トレース学習支援システム不使用下でのトレース実行状況

学習者	課題1	課題2	課題3	課題4	課題5	課題6	課題7	課題8	課題9	課題10	課題11	課題12	課題13	課題14	課題15	課題16
a	3	2	3	15	9	0	2	10	6	18	19	18	21	5	23	9
b	3	2	3	0	1	5	15	0	9	14	27	6	12	5	14	11
c	3	2	2	0	17	23	21	3	1	3	25	10	1	1	5	2
d	3	1	6	10	11	4	8	4	8	3	8	17	5	24	8	17
e	0	4	2	30	0	4	7	4	8	5	2	0	3	2	6	17
f	3	2	3	16	11	7	7	8	5	13	**16**	17	20	3	20	**33**
g	2	1	1	16	9	11	5	6	2	4	19	9	10	19	21	33

指導によるトレース実行あり
学習者独自のトレース実行あり
内の数値は、翻訳プログラム数

5.2　効果的なトレース教育の方法

学習者は行き詰まり状態で自らが構築技術いわゆる戦術を獲得することも多いが，前節の結果からもプログラミング教育の初期段階で個別指導もなく口頭での指導だけで学習者自らがトレースの使用を開始することは稀である．したがってトレース学習には口頭による教授だけでなく，学習者の行き詰まり状態で何らかの個別指導が必要である．しかもトレースを確実に学習させるには，プログラムの行き詰まり状態という環境とそこでトレースを実行するという経験が必要である．しかし，学習者のプログラム作成過程は個々により異なり，生成の全過程を教師が短時間に掴むのは簡単ではない．特に学習者主導で行われる実習では，行き詰まり状態で教師の個別指導を受けるか否かの判断も学習者に任され

ている．そのため教師の個人指導でトレース教育の効果を上げるにも限界がある．以上から効果的なトレース教育のためには，プログラミング実習環境にトレース学習を支援する機能が必要である．

6. トレースを指導するデバッグ支援システム

6.1 デバッグ支援システム DESUS の目的

トレース教育は，実際の行き詰まり状態のプログラムを教材にして指導することが最も効果がある．しかし学習者個々人の行き詰まり状態は同時には発生しないし，個々人で行き詰まりの様相も異なる．確実にトレース技術を身につけさせるには，デバッグの行き詰まりの場面で個人指導することが最も効果的であるが，集合的なプログラミング実習で教師の個別指導を徹底させるには限界がある．そのために，本研究では学習者の行き詰まり状態を手掛かりにしてトレースの個別指導をする支援システム DESUS（Debugging Support System）を構築した [46),47)].

デバッグ過程は，3 章の図 3.5 の基本モデルで示したサイクルが繰り返されていると表すことができる.この基本モデルを基にトレース実行を含めたデバッグ過程を表現すると，図 6.1 に示す図となる．図 6.1 の上半分は通常のデバッグ過程を表している．つまり，誤り箇所を訂正するプランを生成し，プログラムを実行する．その結果を評価し，目的の結果が得られない場合，さらにその原因に対する仮説を立てて新しいプランを生成する．デバッグは，このサイクルをすべてのゴールに対するプランが正しいと評価されるまで繰り返す形で進行する．誤りに対する訂正プランが生成される限りこのサイクルは繰り返されるが，プラン生成が困難になると行き詰まり状態となる．初心者は正確なプラン知識が少ないために行き詰まりが起こりやすいと考えられる．この行き詰まりを打開するひとつの方法としてトレースがある．

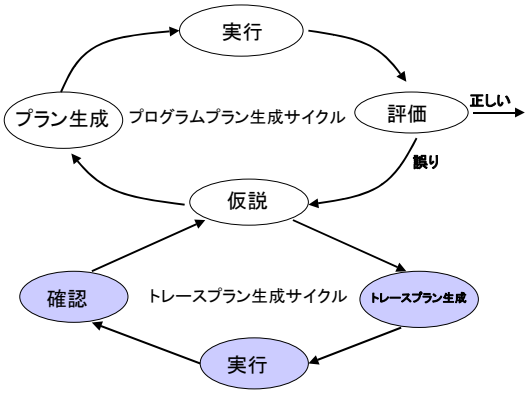


図 6.1　デバッグ過程モデル

トレースは，作成しているプログラムの誤り箇所を仮定して，その仮説に基づき誤りを確定あるいは誤り箇所の調査をする方法である．トレースを使うためには誤り仮説に基づいたトレースプランの生成が必要となる．このトレースプランは，最終的なプログラムには不要なプランであり，この点がプログラム作成のためのプランとは異なる．このような過程は図 6.1 の下方に示すように，トレースプランの生成，実行，確認のように通常のプログラムプラン生成とは別のサイクルを辿ると考えられる．そこでこれをトレースプラン生成サイクルと呼び，通常のプログラムプラン生成サイクルと区別して示す．トレースプラン生成サイクルは，プログラムの誤り箇所を推定することから行き詰まり状態のプログラムの動きを変えて別の角度から自分のプログラムの動きを見せてくれる．それにより学習者の論理的思考が刺激され，デバッグの行き詰まり状態で新しいプログラムプランを発見することに繋がる [48]．

以上から，DESUS の支援目的は，学習者が行き詰まり状態の時に，このトレースプラン生成サイクルを起動させるための指導をし，結果的にプログラムプラン生成サイクルを活性化させ，デバッグを支援することである．具体的には，学習者にトレースプラン生成サイクルが存在することを教え，その実行を試みさせ，学習者自らがトレース生成サイクルを起動することができるようになることを目的にしている．

DESUS は学習者の行き詰まり状態でトレースを指導するため，トレース指導に必要な教材が学習者自身のプログラムで準備される．そのためトレース技術獲得がしやすい状況で指導することができる．

6.2 DESUS のシステム構成

DESUS は，プログラムの入力，編集，翻訳実行，保存などプログラム開発環境として必要な基本的な機能を備えている．それに加えて次の 3 つの支援機能を備えている．

1. トレースを支援する機能
2. 初心者が最も多く起こす翻訳エラーの解説機能
3. 初心者が最も多く使用する出力命令の解説機能

以下の節では，トレース支援機能を中心にシステムの詳細を説明する．

6.2.1 インターフェース

支援システムを起動すると，図 6.2 に示すウインドウが表示される．これが DESUS

と学習者とのインターフェースとなり，学習者はこれを介してプログラムの開発作業を行う．DESUS の４つウインドウには，次のような役割がある．

・ 編集画面（左上側）

プログラムの入力や編集をする画面である．学習者は，この画面に表示されたプログラムに対して支援を受ける．

・ 出力画面（左中側）

プログラムの実行結果が表示される画面である．翻訳エラーや実行時エラーなどのシステムからのメッセージはこの画面に赤字で表示される．

・ 入力画面（下側）

プログラムから要求されるデータを入力する画面である．

・ 支援画面（右側）

デバッグ支援を受けた場合，DESUS からの支援内容が表示される画面である．学習者はこの画面に表示される指導書に基づき，編集画面のプログラムを変更しトレースを実行する．

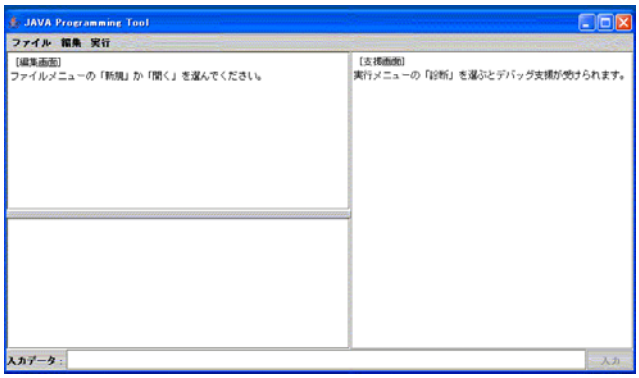


図 6.2　DESUS のインターフェース

6.2.2　操作機能

DESUS の操作は，ウインドウの上部バーにあるファイル，編集，実行の３つのメニューから選択することにより行われる．これを主メニューとして表 6.1 の 11 個の操作が可能である．ファイルメニューからは，プログラムファイルの作成と更新に関して，新規，開く，閉じる，上書き保存，名前を付けて保存の 5 つの機能がある．編集メニューには，エディタとして必要なコピー，カット，ペーストの３つの機能を備えている．実行メニュー

には，実行，支援，強制終了の３つの機能がある．デバッグ支援を求める場合は，実行メニューの中の支援を選択することにより指導が開始される．

表 6.1　DESUS の操作メニュー

主メニュー	副メニュー
ファイル	新規
	開く
	閉じる
	上書き保存
	名前を付けて保存
編集	コピー
	カット
	ペースト
実行	実行
	支援
	強制終了

6.2.3　支援システムの機能

6.2.3.1 学習者との会話手順

DESUS と学習者との会話手順を図 6.3 に示す．DESUS の支援は，学習者の求めに応じて開始される． DESUS は支援を求められると最初に現在のプログラムの状態を学習者に訊ねる．DESUS は学習者の回答を基にトレース指導方法を決定する．次に DESUS が決めたトレース方法に複数のトレース対象がありうる場合，学習者に対象を絞るように求める．ここで得られた情報を基に，DESUS は指導書を作成し支援画面に表示する．これにより学習者にトレースを実行することを求める．

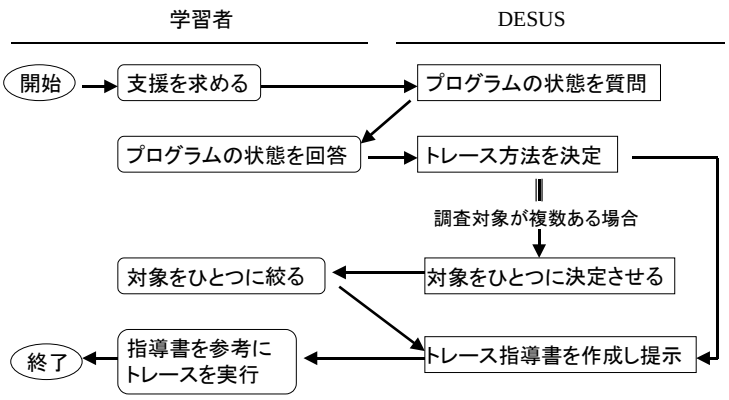


図 6.3　DESUS と学習者との会話手順

6.2.3.2 支援システムの処理手順

DESUS は，6.2.2 操作機能で示したメニューに対応する処理モジュールで構成されている．開発言語は Java である．DESUS の機能のうち，支援に関するモジュールの処理手順を図 6.4 に示す．DESUS は，支援を開始すると「指導方法を決める会話」で学習者にいくつかの質問をする．その内容からトレースの指導方法を決定し，決められたトレース指導方法に必要な情報を「指導に必要な情報収集のための会話」で学習者に尋ねる．この 2 種類の会話で得られた情報と編集画面に表示された学習者のプログラムをもとに指導書を作成する．作成したトレース指導書を支援画面に提示することで支援を終了する．

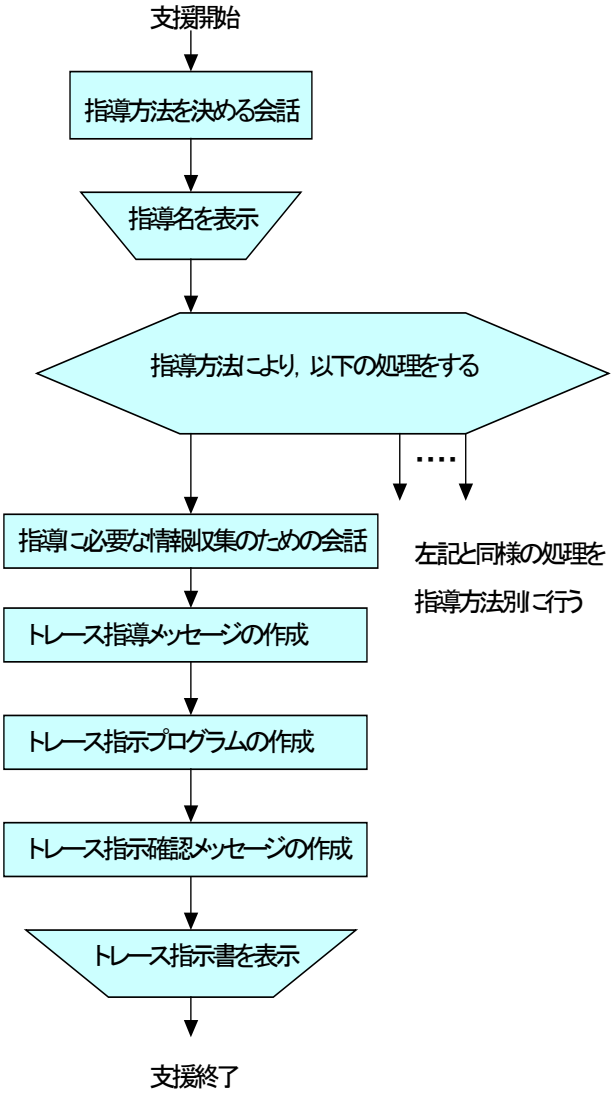


図 6.4　DESUS の処理手順

6.2.3.3 指導方法を決める会話

DESUS は，学習者プログラムの誤りを自動的に診断する分析機能は備えていない．初心者プログラムの誤り診断に関しては，INTELTUTER[2]，PROUST[4]，MARCEL[6]などの研究例があるが，いずれも特定の課題やアルゴリズムに特化したシステムであり，実践教育で与えられる課題すべてに対応できる診断システムではない．DESUS でこれを実現するには，システム構築の負荷が極端に増大するうえに，学習者に提示する課題すべてに対応できるシステムを開発するには，現状ではシステム構築の負荷以上の困難さがある．そのため，DESUS では図 6.5 に示す質問応答グラフを用いて学習者の誤りを推定し，トレース指導方法を決定する．図 6.5 の質問で決定される指導方法は 11 通りある．その指導の内容については，6.2.3.5 トレース指導の内容で詳細に述べる．

デバッグ時に学習者が支援を求めた際に教師は学習者のプログラムについていくつかの質問をする．図 6.5 の質問推移は，この時の教師の質問を想定して構成されている．こ

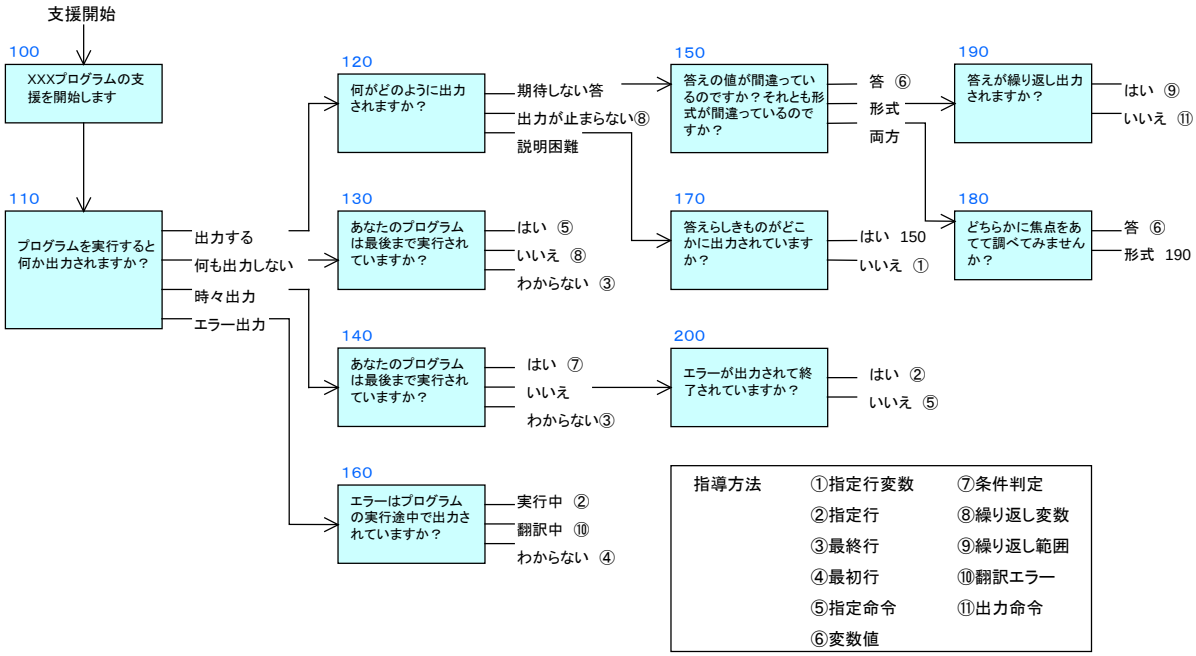


図 6.5　トレース指導方法を決める一連の質問推移

れらはデバッグ時にプログラム作成者が把握しておかなければいけない情報であり，これを明確にすることがデバッグの第一歩である．したがって，この会話はデバッグ時に把握する必要のある要点を学習者へ教えるための指導的会話でもある[49]．そのため，学習者が質問に答えられない，あるいは正しく答えられないことによるトレース指示への齟齬が起こる可能性はあるが，本研究では質問への回答を強いることによる教育的側面を重視して，この方法を採用した．

6.2.3.4 トレース対象を絞るための会話

指導方法が決まった後に，DESUS は“プログラムの終了確認のトレースを指導します”のように指導方法を示し，学習者の了解を得て次に進む．ここで DESUS は，決定された指導方法に必要な調査対象を絞るための質問を学習者にする．例えば，調査対象となる変数名や命令が複数ある場合，変数名や命令の位置などを学習者に選択させる．したがって，指導方法によりこの質問は異なる．図 6.6 に調査対象を絞るための会話を示す．この図に示されない「最初行」，「最終行」，「指定行」の指導は，特に対象を絞る必要がないためこれらの会話をせずに指導書を作成する．

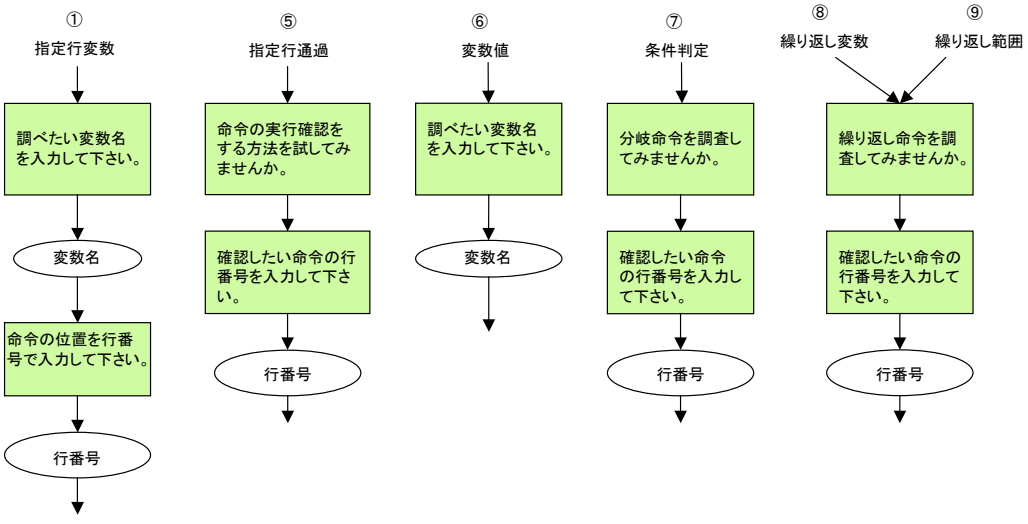


図 6.6　調査対象を絞るための質問

調査対象を絞るための質問には，学習者の基礎的なプログラミング知識が試される内容が含まれている．例えば，図 6.5 の質問で DESUS が繰り返し処理に不都合が生じている可能性があると判断した場合，この調査対象を絞るための質問で「使っている繰り返し命令のうち，調べてみたい繰り返し命令のある行番号を入力して下さい」と質問する．学習者は，自分のプログラムの中で使用されている繰り返し命令を答えなければいけない．これらが複数ある場合は，それらのうちのひとつを調査対象に絞る必要がある．プログラミングの初心者には，これらの質問で基礎的なプログラミング知識を試されるとともに誤りの仮説を立てることも迫られる．このように，DESUS の質問はトレース指導方法の決定や指導書作成のための情報収集の役割とともに，トレースを使用する際の手順を学習者に指導する役割もある．

6.2.3.5 トレース指導の内容

DESUS の指導は，大きく分けてトレース指導と翻訳エラーの解説，出力命令の解説の 3 つに分類される．このうちトレース指導に関するものが最も多く 9 通りある．表 6.2 に DESUS が指導する内容を分類したものを示す．指導名左側の番号欄には，図 6.5 で示された指導方法の番号を示す．

表 6.2　DESUS の指導内容

種類		番号	指導名	指導内容
トレース指導	位置通過	①	指定行変数	指定した命令実行時の変数内容を確認
		②	指定行	実行時エラーの発生箇所を確認
		③	最終行	プログラムが最終まで実行されているかの確認
		④	最初行	翻訳エラーか実行時エラーかの確認
		⑤	指定行通過	指定した命令が実行されているか否かを確認
	変数推移	⑥	変数値	指定した変数の実行時の推移を確認
	条件判定	⑦	If	if 命令の実行状況の確認
			Try	try 命令の実行状況の確認
			Switch	switch 命令の実行状況の確認
	繰り返し	⑧	For 変数	for 命令の実行状況を条件変数の推移で確認
			While 変数	while 命令の実行状況を条件変数の推移で確認
		⑨	For 範囲	for 命令の実行範囲の確認
			While 範囲	while 命令の実行範囲の確認
エラー説明		⑩	翻訳エラー	翻訳エラーの見方と訂正のアドバイス
命令解説		⑪	出力命令	print および println メソッドの使い方の解説

「位置通過」に分類されるトレースには,「指定行変数」,「指定行」,「最終行」,「最初行」,「指定行通過」の 5 通りの指導方法がある．このうち「指定行」は，実行時エラーの発生箇所を調査するためのトレースを指導する．それに対して,「指定行通過」は，調査する命令の位置を学習者に指定させることにより，その命令が実行されているか否かをトレースで調査することを指導する.「最初行」は，エラーが翻訳エラーか実行時エラーかの判別ができないと答えた学習者に，それを確認する方法としてプログラムの最初の位置に“start of program”というメッセージを表示するトレースを指導する．同様に,「最終行」は，自分のプログラムが正常に終了しているか否かが分からないと答えた学習者に“end of program”のメッセージをプログラムの終了箇所で表示するトレースを指導する.「位置通過」に分類される指導のうち,「最初行」と「最終行」は，トレース命令を挿入する位置は DESUS が識別し指導書を作成する．その他の「位置通過」に分類される指導は学習者が入力した行番号や変数を使用して指導書を作成する．

「変数推移」に分類される指導は,「変数値」だけである．この指導では，出力される答そのものが間違っていると答えた学習者に答に関係がある変数名を入力させ，学習者のプログラム内でその変数が使用されている箇所すべてにトレースするように指導する．変数が使用されている箇所は DESUS が識別し指導書を作成する．

「条件判定」に分類されるトレースは，プログラム内で使用されている if 命令，try 命令，switch 命令のどれかを調査対象として，その命令で判定された真偽を確認するトレースを指導する．挿入位置は，DESUS が識別し指導書を作成する．具体的には，学習者が答えた条件命令の条件が真の場合に実行される命令の最初にトレース命令を挿入するように指導する．

「繰り返し」に分類されるトレースには，繰り返し変数の推移を調べる「for 変数」と「while 変数」，繰り返し範囲が意図どおりか否かを確認させる「for 範囲」と「while 範囲」の 4 通りがある.「for 変数」と「while 変数」は，繰り返しに使用されている変数と繰り返しブロックの最初の命令位置を DESUS が識別しトレースを指導する．それに対して「for 範囲」と「while 範囲」は，繰り返しブロックの最後にトレース命令を挿入するように指導する．いずれの場合も，トレース命令挿入位置は DESUS が識別する．

6.2.3.6 トレース指導以外の指導

学習者が「何か出力する」が「期待しない値」で「繰り返し表示されることはない」と

答えた場合，可能性のある誤りが多様でトレースによる効果が想定しにくいため，トレース指導ではなく Java 言語の出力命令を解説する．プログラミングの初期段階の学習者は出力命令を最も多く使用する．それにも関わらず必ずしも正確に記述できていない場合がある．また，トレースする場合にも出力命令を使用する．そのため，この機会を利用して出力命令の使い方を再確認させる目的でこの指導方法を設定した．

プログラムを実行すると「エラー出力」し，そのエラーは翻訳中に出力すると答えた学習者には，翻訳エラーの解説をする．ここでは，「次のようなエラーメッセージがありませんか？」と具体的なエラーメッセージを示し，このメッセージの意味と初心者が多く起こす可能性のある誤り原因について解説する．解説するエラーは 5 種類である．これは半期の教育期間に学習者が起こしたすべての翻訳エラーデータから上位 5 種類を選択したものである．図 6.7 に翻訳エラー解説の一部を示す．

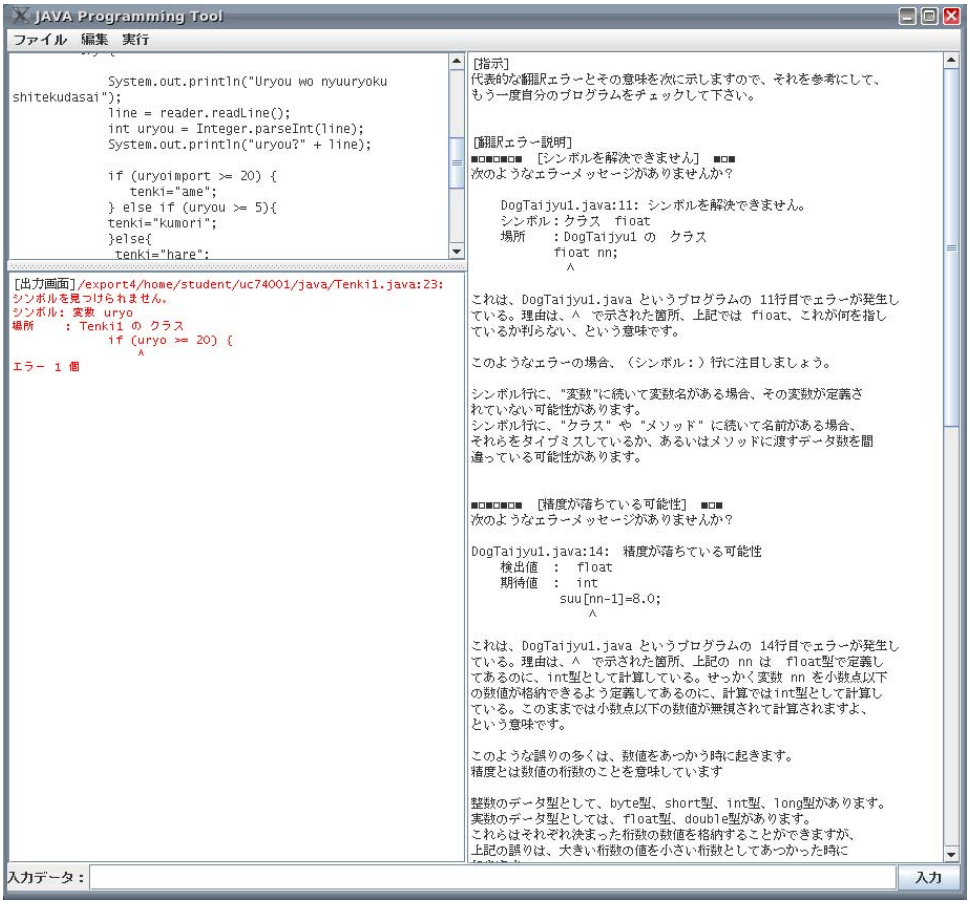


図 6.7　翻訳エラー解説画面の例（一部）

トレース指導と直接関係のない翻訳エラーメッセージ解説と出力命令解説を加えた理由は，利用者である初心者がデバッグで最も困難を感じる翻訳エラーメッセージの意味理解と最も多く使用する出力命令の支援をすることにより，デバッグ支援を補強するとともに，DESUS への親和性を高めることが目的である．

6.2.3.7 トレースの実行

DESUS は，一連の会話から得た情報から指導書を作成し学習者の支援画面に表示するが，提示されたトレースを実際に実行しその結果を確認することは学習者に任せる．学習者はトレースの指示書を読み，それに従ってプログラムを変更し実行することによりトレースを実体験する．この行動がトレース学習には重要であると考えている．また，学習者が指示に従ってトレースを実行した後に，学習者が選択しなかった対象に対して DESUS が別の指示書を作成し指示することはしない．そのため，誤り原因がひとつでない場合，学習者は指導されたトレースを実行してみても行き詰まりが解消しないこともありうる．このような機会にこそ，DESUS で指導されたトレースを参考に自らがトレースを試みることを期待している．

6.2.4　トレース支援の例

6.2.4.1 DESUS との会話例

DESUS へ支援を求めると，学習者のウインドウ上に最初の質問「プログラムを実行すると何が出力されますか」が表示され，質問に対する回答の選択肢もプロンプト内に表示される（図 6.8）．学習者は，この選択肢の中から回答を選択する．その後，図 6.5 に示す会話推移にしたがって質問が提示される．

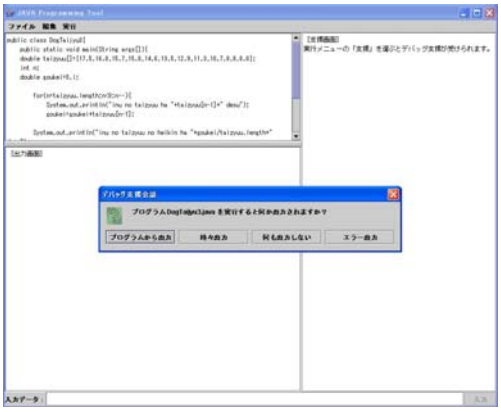


図 6.8　DESUS からの最初の質問

図 6.9　指導内容を知らせる画面

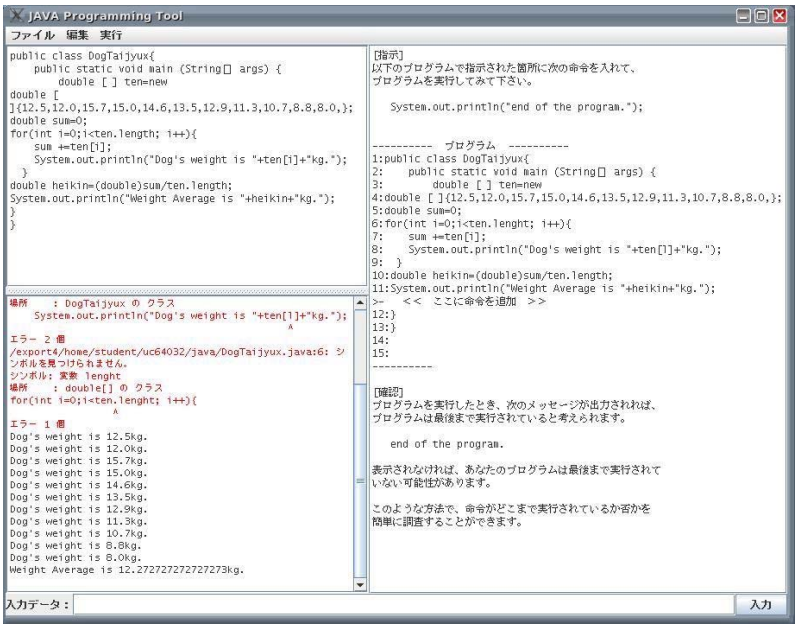


図 6.10　「最終行」指導書提示の画面例

これら一連の質問に回答することでひとつの指導方針が決定され，その指導名が図 6.9 のように表示される．学習者がこれを了解することによってその指導が開始される．図 6.9 の「最終行」指導の例では，「プログラム終了確認のトレースを指導します」と表示され，了解すると図 6.10 に示すトレース指導書が表示される．

「最初行」,「最終行」,「指定行」,「翻訳エラー」,「出力命令」以外の指導内容の場合は，指導内容を知らせる画面（図 6.9）の後に対象を絞るための質問が開始される．図 6.11 は

変数名を入力することが求められた画面例である．入力された変数名や行番号は DESUS により検証される．入力された値に該当する変数や行番号がない，あるいは行番号に相当する命令がないなどの場合はエラーメッセージが表示され指導書は表示されない．例えば学習者が指定した変数が使われていない場合，「変数 n に該当する変数が見つかりません．もう一度調べてみて下さい」と表示される．あるいは学習者が指示した位置に所定の命令がない場合は，例えば「23 行目には分岐命令が見つかりません．行番号 23 が正しくない可能性があります．もう一度調べてみて下さい．」のようにメッセージを表示する．

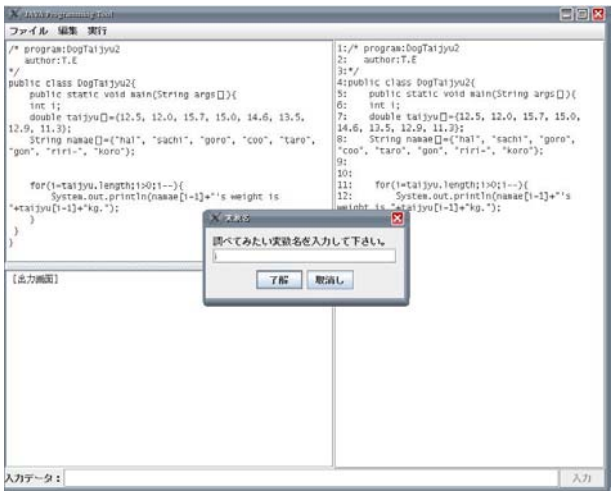


図 6.11　変数名入力指示プロンプトの例

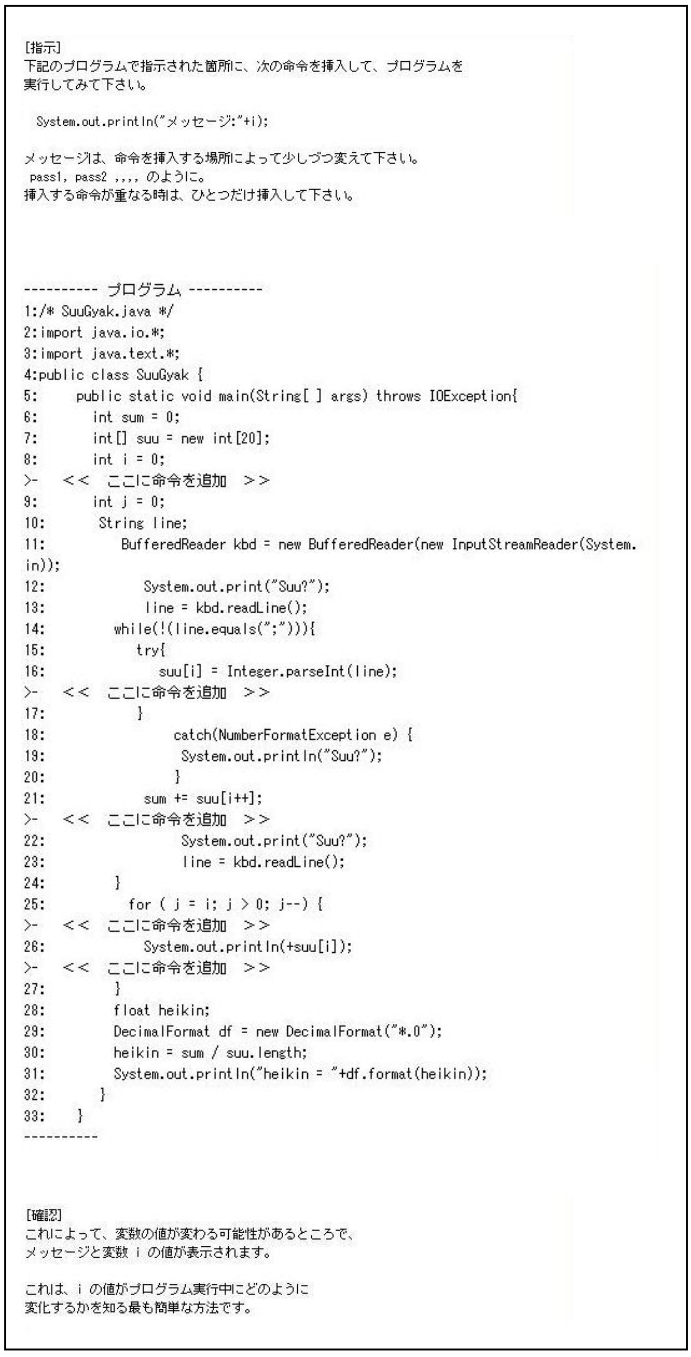
6.2.4.2 トレース指導書の例

DESUS のトレース指導書は，［指示]，[プログラム]，[確認]の 3 種類の内容で構成されている．[指示]には，主としてトレースのために挿入する命令を示す．[プログラム]には，支援を求めた時点の学習者のプログラムを用いて，トレース命令を挿入する位置を指示する．[確認]では，指示したトレースを実行することによって確認できる内容を示す．図 6.12 は「変数値」トレースの指導書の実例である．この例では，学習者が変数名 i を調査対象に指定したために，[指示]で次のトレース命令を使用するように指示している．

System.out.println(“メッセージ：”+i)；

[プログラム]には，“＜＜ここに命令を追加＞＞”と変数が使用されている 5 箇所を検出しトレース命令を挿入するように指示している．学習者のプログラムを変更せずに命令挿入位置を指示するために，“＜＜上の行の｝の前に命令を追加＞＞”などのような指示をす

ることもある．さらに[確認]事項として，「これによって，変数の値が変わる可能性があるところで，メッセージと変数 i の値が表示されます．これは，i の値がプログラム実行中にどのように変化するかを知る最も簡単な方法です.」とトレースの見方と目的を示している．



```
[指示]
下記のプログラムで指示された箇所に、次の命令を挿入して、プログラムを
実行してみて下さい。

  System.out.println("メッセージ:"+i);

メッセージは、命令を挿入する場所によって少しづつ変えて下さい。
 pass1, pass2 ,,,, のように。
挿入する命令が重なる時は、ひとつだけ挿入して下さい。



---------- プログラム ----------
1:/* SuuGyak.java */
2:import java.io.*;
3:import java.text.*;
4:public class SuuGyak {
5:     public static void main(String[ ] args) throws IOException{
6:       int sum = 0;
7:       int[] suu = new int[20];
8:       int i = 0;
>-   <<  ここに命令を追加  >>
9:       int j = 0;
10:       String line;
11:          BufferedReader kbd = new BufferedReader(new InputStreamReader(System.
in));
12:             System.out.print("Suu?");
13:             line = kbd.readLine();
14:         while(!(line.equals(";"))){
15:            try{
16:               suu[i] = Integer.parseInt(line);
>-   <<  ここに命令を追加  >>
17:            }
18:                 catch(NumberFormatException e) {
19:                  System.out.println("Suu?");
20:                 }
21:             sum += suu[i++];
>-   <<  ここに命令を追加  >>
22:                  System.out.print("Suu?");
23:                  line = kbd.readLine();
24:         }
25:           for ( j = i; j > 0; j--) {
>-   <<  ここに命令を追加  >>
26:             System.out.println(+suu[i]);
>-   <<  ここに命令を追加  >>
27:         }
28:         float heikin;
29:         DecimalFormat df = new DecimalFormat("*.0");
30:         heikin = sum / suu.length;
31:         System.out.println("heikin = "+df.format(heikin));
32:       }
33:    }
----------



[確認]
これによって、変数の値が変わる可能性があるところで、
メッセージと変数 i の値が表示されます。

これは、i の値がプログラム実行中にどのように
変化するかを知る最も簡単な方法です。
```

図 6.12　「変数値」トレースの指導例

7　DESUS を用いたプログラミング教育の実験と評価

7.1　実験方法

DESUS によるトレース指導の効果を確認するために，2006 年度および 2007 年度の後期に，宇部フロンティア大学短期大学部の情報システム学科で選択科目として開講されたプログラミングに関する講義・演習科目で DESUS を用いて教育した．講義・演習課目は連続した 2 コマ（1 コマ 90 分）で週に 1 回開講され，受講生の合計は 32 名（2006 年度 27 名，2007 年度 5 名）であった．但し，分析対象の学習者としては，提出を求めた課題のうち 5 課題以上が未作成の受講生 7 名を排除した 25 名（2006 年度 21 名，2007 年度 4 名）を対象とした．学習者は全員この科目で始めてプログラミングを学習した初心者である．

表 7.1　DESUS 評価のためのプログラミング学習項目

学習項目 ＼ 課題	基本構造	演算	変数／データ型	データ入力	入力例外処理	条件判定処理	繰り返し処理	メソッド	配列操作
課題 1	●								
課題 2	●	●							
課題 3	●	●							
課題 4	●	●	●						
課題 5	●	●	●	●	●				
課題 6	●	●	●	●	●				
課題 7	●		●	●	●	●			
課題 8	●	●	●	●	●	●			
課題 9	●		●	●	●	●			
課題 10	●	●	●	●	●		●		
課題 11	●	●	●			●	●		
課題 12	●	●	●	●	●	●	●		
課題 13	●	●	●	●	●	●	●		
課題 14	●	●	●				●	●	
課題 15	●	●	●	●	●	●	●	●	
課題 16	●	●	●				●		●
課題 17	●	●	●				●		●
課題 18	●		●	●	●		●		●
課題 19	●	●	●	●	●	●	●	●	●

● ：課題作成に必須項目

：既学習項目

演習では，表 7.1 に示す 19 個の課題を課題番号順に提示した．このうち 2 課題は，2006 年度と 2007 年度で内容が異なるが，学習目標や難易度は同程度であるため，特に区別をしていない．表 7.1 では，各課題の主たる学習項目を●印で示し，各課題作成時の既学習項目の背景を灰色で示している．この内容からも明らかなように，この教科目の目標は，プログラミングの基本項目（変数，データ型，基本構造など）を理解し，これらを用いた基礎的なプログラムが作成できることである．

学習者には，演習に先立ち DESUS 環境下でのプログラム開発方法と支援の受け方について説明し，「課題 1：イニシャル表示」の作成を通して DESUS の使用方法を指導した．さらに DESUS の支援は，プログラムの誤りを発見してくれるのではなく，トレースという誤りを見つけるための方法を指導してくれることを口頭で伝え，教育期間中にトレース方法を一斉に教授することはしなかった．

7.2 分析データ

7.2.1 学習者のプログラム

学習者が課題プログラムを作成中に翻訳したプログラムをすべて自動的に保存した．この実験中に保存された被験者 25 名分のプログラム総数は，12262 本（2006 年度 10202 本，2007 年度 2060 本）であった．これらのプログラムを各学習者の課題別に抽出し，トレース使用の可能性の有無が判断できる資料作成に利用した．また，これらのプログラムを時系列に翻訳実行し，学習者の辿ったデバッグ過程を再現することによって学習者別課題別にトレース実行過程データを作成した．

7.2.2 DESUS 使用ログデータ

学習者が DESUS に支援を求めると，学習者のプログラム名，トレース指導名，日時が学習者別に保存される．この情報は，学習者の DESUS 利用状況を集計し，学習者のトレース実行時の状況を解析するために利用した．

7.2.3 筆記試験結果

演習最終日に 90 分間の筆記試験を行った．2006 年度 2008 年度ともに同じ内容の試験を同じ条件で行った．問題は 5 種類のプログラムを提示して，各プログラムに関して 2 問から 5 問の回答を求めるものである（資料 1）．評価は 100 点満点で採点した．

7.2.4　課題別トレース実行過程データ

トレース実行が行われた可能性がある箇所を選択しやすくするために，学習者のプログラムで使用されている出力命令を課題プログラム毎に時系列に自動抽出した資料を作成した．この資料と DESUS 使用ログデータを参考にして，トレースを実行した可能性がある学習者のデバッグ過程を再現しながら，トレース実行の有無やトレース実行時のプログラムの状態などを記録した．

表 7.2　学習者別 DESUS 使用頻度とトレース実行状況

学習者	課題1	課題2	課題3	課題4	課題5	課題6	課題7	課題8	課題9	課題10	課題11	課題12	課題13	課題14	課題15	課題16	課題17	課題18	課題19	支援回数	支援課題数
1						1												1		2	2
2											1					1		1	x	3	3
3					3		7	2		2		1						2	2	19	7
4																			x	0	0
5	x																	1		1	1
6					6				1		1	2					1			11	5
7																		1		1	1
8							1	1												2	2
9					1														x	1	1
10																			x	0	0
11								1		1			1	2	x	2			1	8	6
12		1					2			1		1	1					4	4	14	7
13					1			1		1	2									5	4
14													1							1	1
15								1							1			1		3	3
16										3										3	1
		x		2	1	2				2		4	4			1			1	17	8
18						1										1			x	2	2
19	x						1	2	1	1			2				2			9	6
20					1			3											2	6	3
21										1		x			x			x	x	1	1
22						1			4		1						1	x	2	9	5
23													2	x				x	x	2	1
24				1					4	1	1							4	1	12	6
25								1										x	1	2	2
																			Total	134	78
																			Average	5	3

x	課題未作成
2	DESUS が2回トレース支援
	DESUS の指示でトレース
	DESUS の指示なしでトレース

7.3　分析

7.3.1　トレース実行に関する分析

各学習者の DESUS 使用ログデータと課題別トレース実行過程データをもとに，学習者のトレース実行状況を表したものが表 7.2 である．表 7.2 の数値は，DESUS がトレース指導した回数を示している．x で示される箇所は，学習者が課題を作成していない箇所である．また，薄い背景色の箇所は，DESUS の指示に基づきトレースを実行した箇所であり，濃い背景色の箇所は，DESUS の指示を受けずに学習者が独自にトレースを実行した形跡がある箇所である．ひとつの課題で DESUS の指示によるトレースと学習者独自のトレースの両方がある場合は，学習者独自のトレースを優先させ濃い背景色で表している．

DESUS からのトレース指導を全く受けていない学習者 2 名を除く 23 名（92%）が最大 19 回，平均 5 回の指導を受けていた．学習者 25 人中 13 人（52%）がトレースを実行していた．このうち 9 名（36%）の学習者は，DESUS の支援を受けずにトレースを実行した形跡が確認され，これらの学習者はトレース技術を習得していると判断した．したがってトレースを学習した割合は 36%で，DESUS を使用していない 2003 年度の 14%に比べその割合は多くなっている．

DESUS の支援なしにトレースを実行していた 9 名の学習者は，トレース技術を獲得しており，彼らをトレース学習者と呼ぶ．DESUS の支援を受けてトレースを実行しているが, 独自にトレースを実行した形跡がない 4 名の学習者をトレース実行者と呼ぶ. DESUS を使用しているか否かに関わらずトレースを全く実行していない 12 名の学習者をトレース未実行者と呼ぶ．これらのトレース実行段階別に DESUS の使用とトレース実行の平均値を示したのが表 7.3 である．

表 7.3　トレース実行段階別平均実行回数

トレース実行段階	平均値		
	DESUS 支援回数	トレース実行回数	1 課題の翻訳回数（標準偏差）
トレース学習者	9.7	5.7	34（ 8.4）
トレース実行者	3.8	1.3	25（13.0）
トレース未実行者	2.7	0.0	20（ 9.3）

表 7.3 には，ひとつの課題プログラム完成までに要した平均翻訳回数も掲載している．ここではトレース学習者が最も多い平均 34 回の翻訳実行を行って課題を完成しているのに対して，トレース実行者やトレース未実行者は 25 回，20 回と少なくなっている．トレースは，デバッグ時にプログラムの状態を把握し行き詰まり状態を脱出する方法のひとつであるため，この技術を獲得した者は行き詰まり状態から早めに脱出し，むしろ少ない翻訳実行回数でプログラムを完成させると予想される．しかし，今回の実験では，トレース学習した学習者グループの方が翻訳実行回数の平均値が高いという結果であった．トレース技術を獲得したことがデバッグ過程を短くすることには繋がっておらず，むしろ長くなっている．

プログラミング学習の初期段階では，学習したプログラミング言語やアルゴリズムなどをプログラムとして実際に組み立てることによって，その際に起こる命令相互の干渉やバリエーションなどを実習で課題プログラムを作成しながら学習する．したがって学習者のデバッグ過程は，プログラムが正しいと確認するだけの実習ではなく，様々なプログラムプランを組み合わせたり新たなプランを試したりする，最も重要なプログラミングの学びの過程でもある．このデバッグ過程でトレースは，学習者のプログラムに新しい現象を意図的に引き起こす．それが学習者の新たな学習の機会を与えることに繋がり，結果として翻訳実行回数が多くなると考えられる．

トレース未実行者が試行錯誤の少ないデバッグ過程で課題プログラムを完成させたことに対しては２つの理由が考えられる．ひとつは，プログラミング適性に優れた学習者の場合，トレースなどをするまでもなく入門程度の課題は自力で作成ができる．もうひとつは，ほぼ完成したプログラムを正しく動くことを確認するだけで試行錯誤を回避している場合である．

以下では，トレース実行段階別に次のような視点から分析を行う．トレース学習者の分析では，トレース学習者は DESUS の指導によりトレース技術を獲得したのか，トレース実行者の分析では，なぜトレース実行者は独自のトレースを実行しなかったのか，トレース未実行者の分析では，なぜトレース未実行者はトレースを実行しなかったのか，の視点から DESUS の評価を行う．

7.3.2　トレース学習者の分析

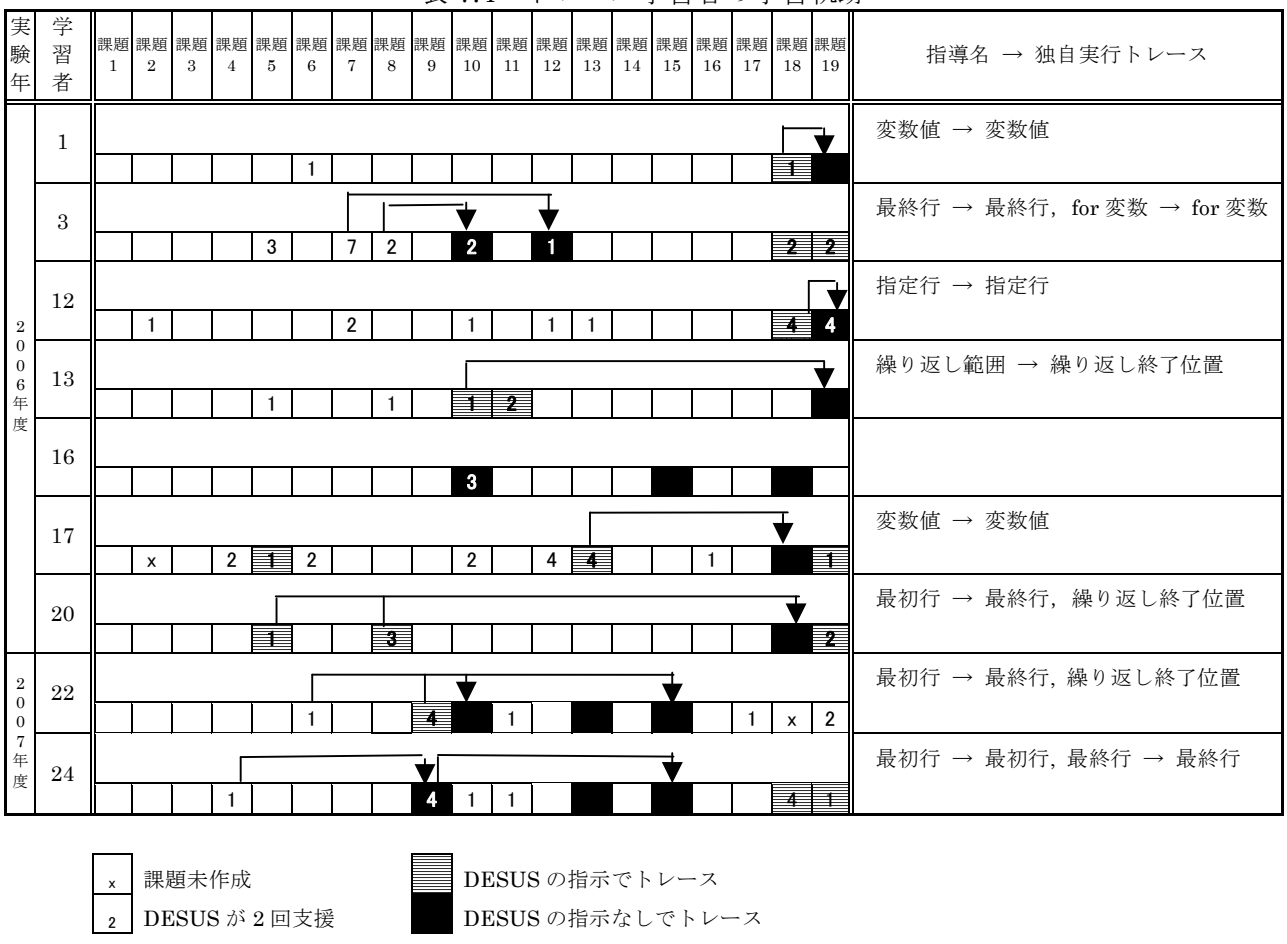
7.3.2.1 トレース学習者の学習過程

トレース学習者 9 名のトレース学習過程を表 7.4 に示す．この表は，表 7.2 からトレース学習者の行だけを抜粋したものに，各学習者が独自にトレースを実行する際に DESUS の支援が影響したと考えられる推移を矢印（→）で示している．また学習者が DESUS から受けた指導名と独自に実行したトレース名を左側に示す．

この表から，トレース学習者 9 名のうち学習者 16 を除く 8 名は，DESUS の指導でトレースを学習したと断定した．学習者 16 は，課題 10 で DESUS からの支援を受けているが，その支援を受ける前に while 命令の働きを確認するための特別なプログラムを作成し，独自にトレースを実行していた．したがって DESUS によりトレースを学習したのは，8 名（32%）である．その根拠を以下に述べる．

学習者 1 は，課題 18 で「変数値」の支援を受けトレースを実行した．その直後の課題 19 で変数値トレースを独自に試みている．同様に，学習者 3 は「最終行」と「for 変数」，学習者 12 は「指定行」，学習者 13 は「繰り返し範囲」，学習者 17 は「変数値」，学習者

表 7.4　トレース学習者の学習軌跡



24 は「最終行」と「最初行」の支援を受けた後に同様のトレースを独自に実行していた．このうち学習者 13 は，課題 10 で，DESUS から繰り返し実行する命令の最後の位置にトレース命令を挿入するように指導されたが，繰り返し命令の終了位置にトレース命令を挿入し実行していた．その後，課題 19 で独自にトレースを実行したのは，以前実行したことのある繰り返し命令の終了位置に命令を挿入して実行していた．

学習者 20 と学習者 22 は，どちらも「最初行」の指導を受けていたが，独自に実行したトレースはプログラムの最初ではなく，繰り返し命令の最後あるいはプログラムの最後を確認するトレースであった．「最初行」の指導では，以下のような確認のメッセージを表示している．

> [確認]
> プログラムを実行したとき，“start of the program”が出力された後でエラーが表示されれば，実行時エラーです．
> 何も表示されないでエラーが出る場合は，翻訳エラーです．
> このようにプログラムの最初に，メッセージ出力命令を挿入することによって，プログラムが実行を開始しているか否かを確認することができます．

学習者 20 は，2 つの課題で 1 回づつこの指導を受け，どちらも実行している．学習者 22 は，2 課題で 4 回も続けてこの指導を受け，そのうち 1 回を実行している．したがって，複数回の同じ指導とそれに基づくトレース実行が，その後の課題でプログラムの最後を確認するトレース実行に繋がったと判断した．

7.3.2.2 DESUS 支援によるトレース学習過程の例

DESUS の支援によりトレースを学習した例として，2006 年度の学習者 12 が課題 19 を作成した過程を示す．課題 19 は，キーボードから配列に読み込んだ数値を大きい順に並べ替えて結果をディスプレイに表示する分類の課題である．

学習者 12 の課題 19 のプログラム 19 版は実行時エラーを起こして停止する状態であった．19 版とは，プログラムを作成し始めて 19 番目に翻訳したプログラムを意味している．学習者 12 はここで DESUS に支援を求め，「プログラムからエラーが表示され，そのエラーは実行時に出力される」と報告した．そのため，DESUS は図 7.1 に示す「指定行」を指導した．

[解説]
出力されたエラーメッセージをもう一度良く見てください。
次のような表示がありませんか。

```
 Exception in thread "main"java.lang.Errot:....
           at ........（プログラム名：番号）
```

at ....の行が沢山ある場合は、プログラム名が自分の行を
見て下さい。

これは（プログラム名）の（行番号）の箇所で実行が何らか
の原因で不可能になったということを意味しています。

[指示]
そこで、エラー表示された行番号の前か後ろに、次の命令を
挿入して実行してみて下さい。

```
   System.out.println("メッセージ");
```

```
----------  プログラム  ----------
1:/*program:Sort.java*/
2:/*name:Tomoyuki Tanaka*/
3:/*date:2007.1.25*/
4:
5:import java.io.*;
6:
7:public class Sort{
8:    public static void main(String□arges)throws IOException{
9:    String line;
10:    int□a;
11:    int i=0,t;
12:    a=new int[50];
```

途中省略

```
25:    line=reader.readLine();
26:    }
27:    }
28:
29:
30:    for(int j=0;j<i;j++){
31:      if(a[j]<a[j-1]){
32:      t=a[j];
33:      a[j]=a[j-1];
34:      a[j-1]=t;
35:      }
36:    }
37:
38:    for(int j=0;j>i;j--){
39:      System.out.println((i+1)+"no Suu?"+a[i]);
40:    }
41:  }
42:}
43:
44:
45:
46:
----------
```

[確認]
前に挿入した場合、メッセージが表示されればプログラムは
そこまでは実行されていることになります。
後ろに挿入した場合、そのメッセージが表示されなければ、
その前で実行不可能になっていることになります。

このようにして、プログラムのどこでエラーが発生したかを
調べることを進めます。

図 7.1　学習者 12 の課題 19（第 19 版）での DESUS の指導

```
import java.io.*;

public class Sort{
      public static void main(String[]arges)throws IOException{
      String line;
      int[]a;
      int i=0,t;
      a=new int[50];
BufferedReader reader=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Suuchi wo nyuuryoku shitekudasai");
      System.out.println("owari ha [;]wo nyuuryoku shitekudasai");
      line=reader.readLine();
      while(!(line.equals(";"))){
      System.out.println("Type in?"+line);
      try{

      a[i]=Integer.parseInt(line);
      i++;
      line=reader.readLine();
      }catch(NumberFormatException e){
      System.out.println("tadashiku nyuuryoku shitekudasai");
      line=reader.readLine();
      }
      }


      for(int j=0;j<i;j++){
System.out.println("ppp");   <------- DESUS の支援により挿入

        if(a[j]<a[j+1]){
System.out.println("aaa");   <------- 第 22 版で独自に挿入
        t=a[j];
        a[j]=a[j+1];
        a[j+1]=t;
        }
      }
System.out.println("bbb");   <------- 第 26 版で独自に挿入
      for(int j=0;j<i;j++){
        System.out.println((j+1)+"no Type in?"+a[j]);


      }
System.out.println("sss");     <------- 第 29 版で独自に挿入
   }
}
```

図 7.2　学習者 12 の課題 19（第 29 版）プログラム

学習者は，図 7.1 の指導書に基づき表示されているエラーメッセージから判断して 31 行目にトレース命令を挿入し実行している．その後，第 22 版で，学習者 12 はトレース命令の次にある if 命令の条件式に使われている配列参照を修正し，その if 命令が真の場合に実行される最初の位置に独自のトレース命令を挿入し実行している．その後も第 22 版，

第 29 版で独自のトレースを徐々に追加し，プログラムの実行経過を確認しながらデバッグを進めている．その経過がわかる第 29 版のプログラムを図 7.2 に示す．学習者 12 は，この課題でその後も同様なトレースを 36 版で実行し，68 版，70 版では再び DESUS から「変数名」の指導を受け，第 103 版でこの課題プログラムを完成している．

以上のように，トレース学習したと確認した学習者の多くは，DESUS により指導を受けて実行したトレースと命令の位置関係や出力メッセージなどが類似しており，指導で実行したトレースを模倣するかたちで独自のトレースを試みていた．

7.3.3 トレース実行者の分析

DESUS の支援でトレースの実行を試みた学習者のうち 4 名は独自のトレース実行をしておらず，トレースを学習したと断定できなかった（表 7.2 参照）．学習者は行き詰まり状態で支援を求めるものである．デバッグ時に行き詰まりを感じない場合は支援を求めず，またトレースも実行しないと考えられる．そのため，これらの学習者が DESUS からトレースを指導された後の課題で,プログラムをどのような過程で作成しているかを調査した．このうち学習者 19 を除く 3 名（学習者 14，学習者 15，学習者 18）は，支援によりトレースを実行した後の課題で行き詰まりを感じていないためにトレースを実行する機会がなかったと推定した．その根拠を以下に述べる．

学習者 14 は，課題 14 から課題 19 のデバッグ時に，翻訳エラー以外のプログラム実行回数が 4 回から 8 回で，課題 18 だけが 13 回である．学習者 15 と学習者 18 は，1 回から 6 回の実行でデバッグを終了している．学習者の行き詰まりの感じ方には個人差があるため，この推定には学習者が提出したレポートも参考にした．これら 3 名のそれぞれの課題レポートには，翻訳エラーによる行き詰まりの記載はあるが，いわゆるデバッグ上の行き詰まりを記載したものはなかった．したがって，それらの学習者は DESUS の支援で最後にトレース実行した後にデバッグ上の困難さを感じておらず，トレースを試みる必要がなかったと考えられる．

これらの学習者のうち，デバッグ上の困難さを感じた可能性が多少ある学習者 14 の課題 18 のプログラム作成過程を見てみる．課題 18 は，数値を配列に読み込み，それを逆順に出力する課題である．学習者 14 は，最初に終わりの印が入力されるまで数値を配列に入力するルーティンを,それまでに作成した課題プログラムを参考に完成させた.その後，このプログラムに，配列定義，入力した値を配列に代入する命令，配列のデータを逆に出

力する処理を追加した．つまり戦術として後回し策を用いてプログラムを作成している．命令を徐々に追加して行った際に，次のデータを入力する命令と添え字増加命令が 2 重に存在することになり，バグとなっている．このデバッグに 9 回のプログラム実行を要しており，レポートに「20 個以上入力すると，“エラーメッセージ”（配列参照箇所で添え字範囲を超えて実行したというエラー）が発生した．エラーは，i=i+1; line=reader.readLine(); この部分を 2 回も打っていたのが原因だった」と報告している．学習者は，この実行時エラーを手掛かりにして誤りを発見したため，トレースの必要性を感じなかったのではないかと思われる．

学習者 19 は，DESUS に関して「使い方が分からない」,「どこが間違っているかをもっと具体的に表示して欲しい」と使用するたびにレポートに記載している．つまり DESUS がバグの原因とプログラムの訂正箇所や訂正方法を指示してくれることを期待して使い続けている．このことが DESUS を頻繁に使用しながらもトレース技術獲得に至らなかった最も大きな原因であると推定される．

7.3.4　トレース未実行者の分析

DESUS の支援を受けた学習者のうち 10 名は，DESUS が指示したにも関わらずトレースを 1 度も実行していなかった．これらの原因を分析するために，DESUS により支援を受けた時点のプログラムの状態と指導された内容の適合性を調査した．表 7.5 に，DESUS が指導したトレースを指導名別に集計したものを示す．

表 7.5 には，DESUS の支援回数とともに，その支援で学習者がトレースを実行した回数と未実行の回数，および DESUS が支援した時点の学習者のプログラム実行状態が翻訳エラーであった数を示している．最右列には DESUS の支援数に対して支援を受けた際のプログラムが翻訳エラーを起こすプログラムであった割合を示す．さらに，DESUS の支援に対しトレースを実行しなかったプログラムが翻訳エラーであった割合も示している．

DESUS の支援総数に対して 57%のプログラムが翻訳エラーでトレース指導を受けようとしていた．翻訳エラーを起こしているプログラムで指導を受ける可能性がある「最初行」指導を除くと，55%が翻訳エラーのプログラムで支援を受けていたことになる．

支援を受けたがトレースが実行されなかったプログラムの場合，翻訳エラーを起こしていたプログラムが 74%を占めており，「最初行」指導を除いてもトレース実行がされなかった 70%が翻訳エラーのプログラムでトレース指導を受けていたことになる．

表 7.5　指導別トレース支援内容

指導名	トレース支援回数(A)+(B)	トレース実行(A)	トレース未実行(B)	翻訳エラー時に支援(C)	未実行に占める割合(%) C/B*100	支援数に占める割合(%) C/(A+B)*100
指定行変数	2	0	2	2	100	100
指定行	40	6	34	30	88	75
最終行	7	2	5	1	20	14
最初行	38	10	28	24	86	63
指定行通過	5	1	4	2	50	40
変数値	21	7	14	4	29	19
条件判定	2	0	2	2	100	100
繰り返し変数	16	3	13	11	85	69
繰り返し範囲	3	1	2	1	50	33
翻訳エラー	43	0	43	43	100	100
出力命令	18	0	18	11	61	61
総計	195	30	165	131	79	67
トレース合計	134	30	104	77	74	57

支援回数が多く，しかも翻訳エラー時の支援割合が 75%を超えた「指定行」は，図 7.1 の指導例に示すように，実行時エラーメッセージに表示された位置でエラーが発生していることを確認するトレースである．このような指示を翻訳エラー状態のプログラムで DESUS から受けた学習者は，自分のプログラムに指示されたようなメッセージが表示されておらず指示されたトレース行動を起こせなかったと考えられる．

「指定行」の指導をプログラムが翻訳エラーの状態で受けた学習者は，DESUS からの最初の質問「プログラムを実行すると何が出力されますか」に対して“エラー出力”か“時々出力”を選択している（図 6.5 および図 6.8 参照）．翻訳エラー状態で“エラー出力”を選択した場合，次の質問で実行時エラーであるという“実行中”を選択して「指定行」に辿り着く可能性は少ない．それに対して，最初の質問に“時々出力”と答えた場合，学習者のプログラムはエラーが表示されて終わっていることから，それ以後の質問に対して“プログラムは最後まで実行されておらず”，“エラー表示されて終わっている”と答え，容易に「指定行」指導に辿り着く可能性が高い．

最初の質問の選択肢である“時々出力”は，本来は学習者 14 の課題 18 の作成過程（7.3.3. トレース実行者の分析を参照）の例のように入力データの違いなどで支援対象のプログラムが時に正常に実行されるが時々実行時にエラーが出ることを想定した回答である．しかし，学習者は支援を受けるまでの作成過程を通して，今はエラーが出ているが，以前エラ

ーが出ずに実行されたことを捉えて“時々出力”したと回答した可能性がある．DESUS は学習者のプログラムが実行可能か否かの検査を指導前にしていない．これは DESUS の不備な点であり，そのために翻訳エラーで支援を受けようとした学習者が，トレース指導に従ってトレースを実行できなかったという指導の齟齬が多数発生したと推察できる．

7.3.5　トレース実行とプログラミング学習に関する分析

本研究の最終的な目標は，トレース指導がプログラミング学習に効果的に働くことを目指している．そこで本節では，トレース実行段階とプログラミングの筆記試験結果の関係からトレース学習がプログラミング学習に与えている影響を分析する．

表 7.3 に示したトレース実行段階別の平均値（平均支援回数，平均トレース実行回数，1 課題あたりの翻訳実行回数）を棒グラフで，プログラミングに関する筆記試験の各グループの平均点を折線グラフで示したものが図 7.3 である．トレース学習者の平均点 61.0（標準偏差 21.0）は，トレース実行者の 42.0（標準偏差 20.0），トレース未実行者の平均点 41.8（標準偏差 24.1）に比べおよそ 20 ポイントも良い評価を受けている．トレース実行者とトレース未実行者はほぼ同じ程度の平均点であるが，トレース未実行者の方は個人差が大きい．また，図 7.3 から明らかなことは，トレース実行回数や 1 課題あたりの翻訳実行回数の平均値と試験の平均点がほぼ比例していることである．トレース学習者は，積極的にトレースを実行することでデバッグ時に多くの翻訳実行をしている．それがデバッグ過程を短くすることには繋がっていないが，プログラミング試験による評価は高い結果を出している．

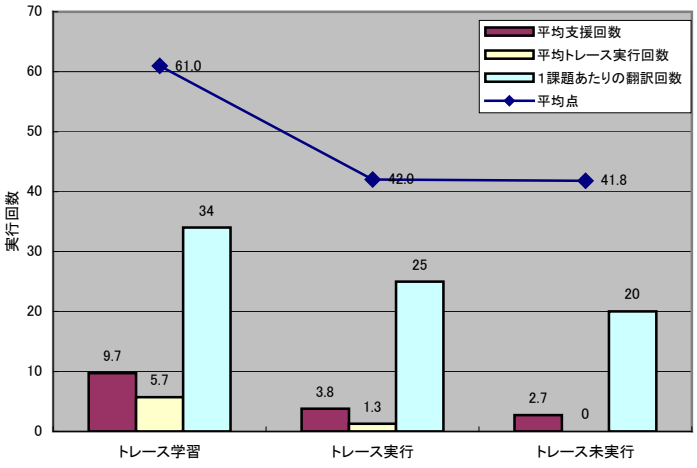


図 7.3　トレース実行段階別平均値

次に，学習者個々のトレース実行回数（支援を受けてトレースを実行した回数と独自のトレース実行を合算した回数）とプログラミング筆記試験結果の関係を図 7.4 に散布図で示す．図 7.4 では，トレース学習者を◇で，トレース実行者を■，トレース未実行者を▲でプロットしている．この図に示すトレース実行回数とプログラミング試験結果には弱い正の相関(r=0.39)があり，トレース学習者は相対的にプログラミング学習も進んでいることが窺える．トレース実行者は，1 名を除きトレース学習者に近い位置にいる．それに対して，一度もトレース実行をしなかったトレース未実行者は，この図からも個人差が非常に大きいことが分かる．個人差が大きい理由として考えられることは，プログラミングの適性が高い学習者は，その推論力を利用してトレースしないまま行き詰まりを解決し，逆に，プログラミング学習が進まない学習者は，DESUS の指導書による支援だけではトレース実行の意味を理解することができず，トレースを実行するまでには至らなかったと考えられる．

次に，翻訳実行回数とトレース実行回数の関係を図 7.5 に示す．トレース実行回数が多い学習者は翻訳実行回数が多くなっている（r=0.53）傾向は，この図からも読み取れる．

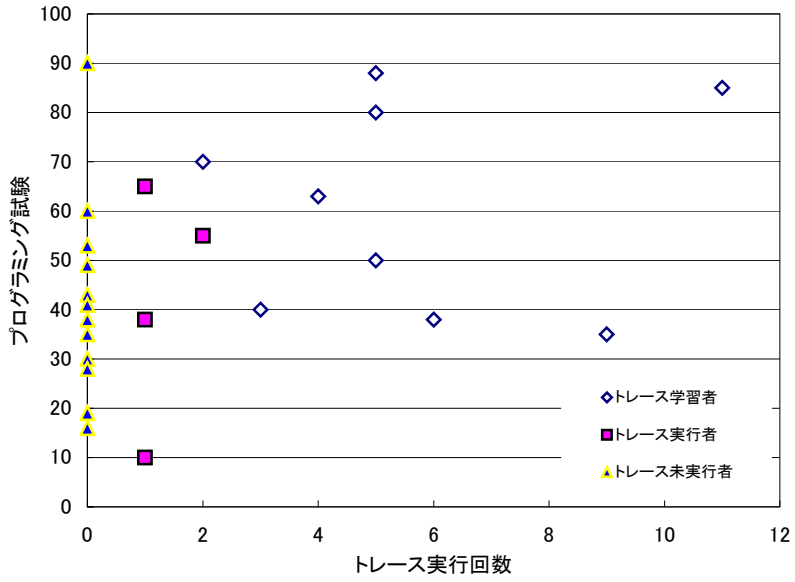


図 7.4　トレース実行回数とプログラミング試験の関係

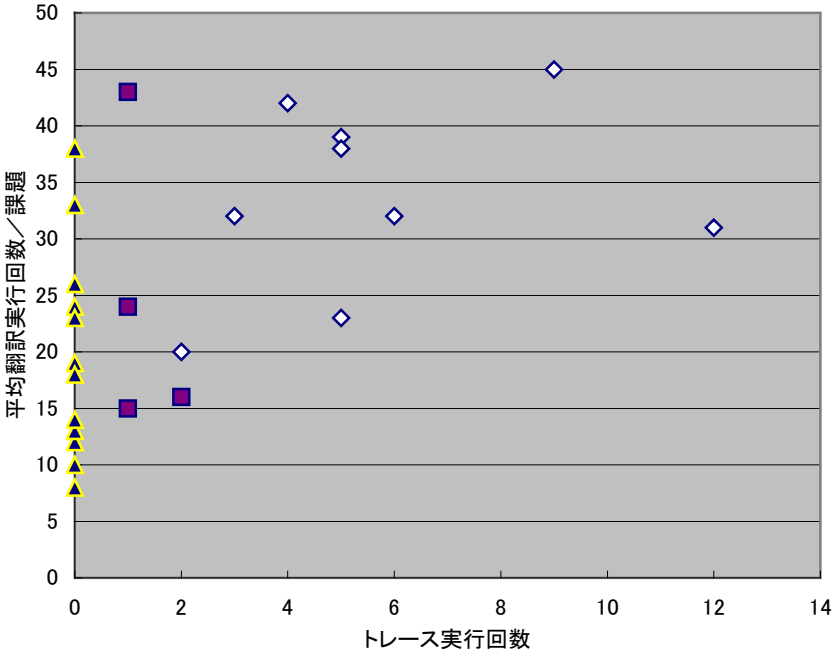


図 7.5　トレース実行回数と翻訳実行回数の関係

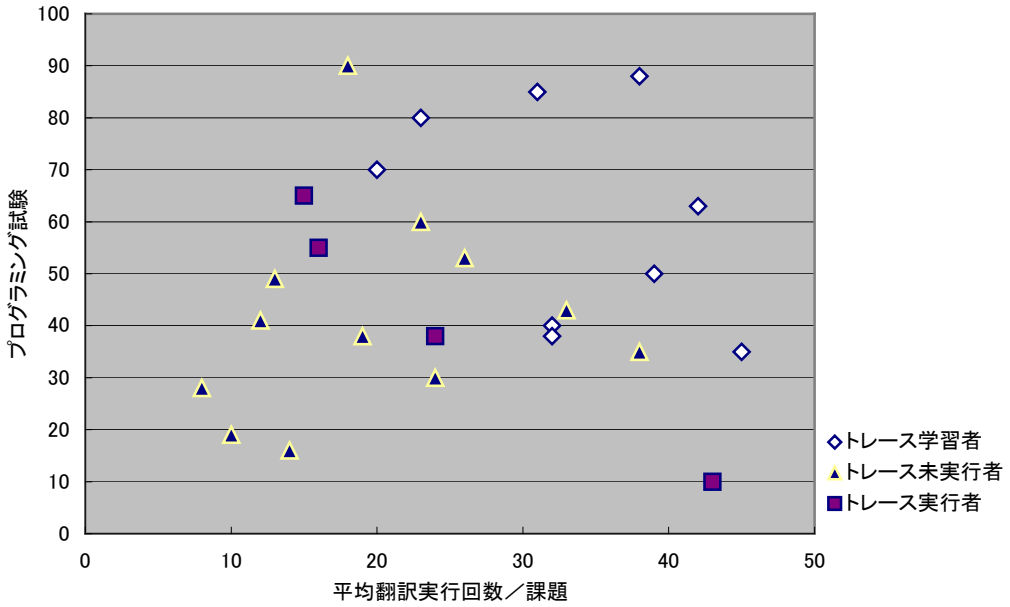


図 7.6　翻訳実行回数とプログラミング試験の関係

図 7.6 に翻訳実行回数とプログラミング試験結果の関係を散布図で示す．被験者すべてを対象にこの両者の相関をみるとほとんど相関(r=0.05)がない．しかし，トレース学習者，トレース実行者のグループに分けてこれらの相関をみると，負の相関がある（トレース学習者の場合 r=-0.38，トレース実行者の場合 r=-0.98）．したがって，トレース学習者やトレース実行者は，トレース未実行者に比べ翻訳実行回数が増加しているが，同じ学習者グループ内でみると翻訳回数の少ない学習者の方が良い評価を得る傾向にある．これらの関係をまとめたものを図 7.7 に示す．

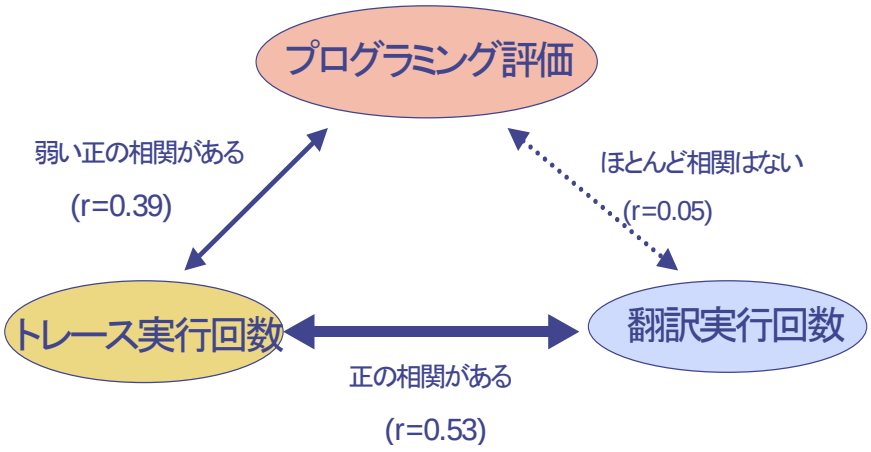


図 7.7　プログラミング学習との相関

7.3.6　トレース支援内容の分析

表 7.5 から，トレース指導に関してだけのデータを抽出しグラフにしたのが図 7.8 である．図 7.8 のトレース実行回数には，学習者が支援なしで実行したトレースは含まれていない．

DESUS が指導したトレースで最も多いのが「指定行」の 40 件である．次に多いのが「最初行」の 38 件である．この 2 つのトレース指導が指導総数 134 件のうちの 58.2%を占めていた．DESUS の支援に対して実際にトレース実行した回数が最も多かったのが「最初行」の 10 件で，「変数値」の 7 件がこれに続いている．最も多く指導した「指定行」はその指導に基づき実際に指導した回数は「最初行」よりも少ない．これは，トレース未実行者の分析で述べたように，翻訳エラーがあるプログラムにトレース指導を求めた割合が 88%もあることが原因と考えられる．

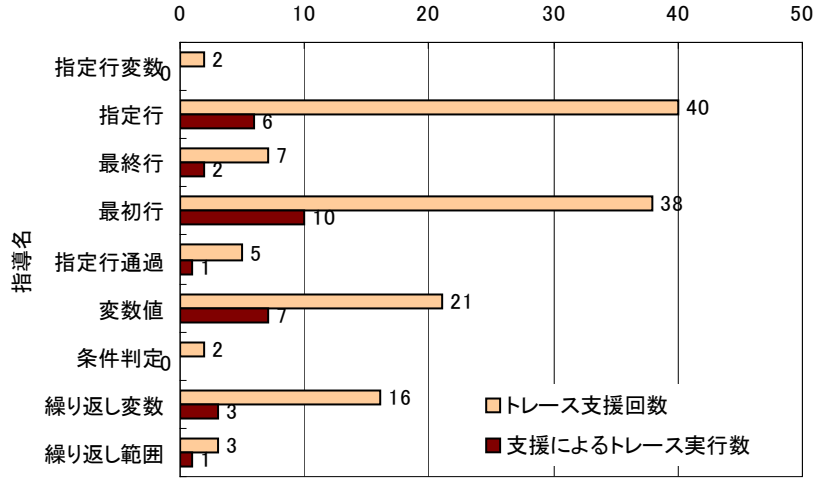


図 7.8　指導内容別トレース支援回数とトレース実行回数

プログラミング言語に習熟していない初期のプログラミング学習段階では，システムから表出されるエラーが識別できないとか，エラーに対する対応が分からないなどの行き詰まりが多い．「最初行」と「指定行」の支援回数とトレース実行数が共に多いのは，このようなプログラミング初心者の特徴が現れていると言える．

7.3.7　トレース指導の開始時期

DESUS のトレース指導回数と学習者のトレース実行回数を課題順にグラフで示したのが図 7.9 である．折線グラフで示した値は DESUS がトレースを支援した回数である．それに対して学習者がトレースを実行した回数は，支援によるトレースと独自のトレースに区別して棒グラフで示した．この図から明らかなように，トレース支援を求める回数は前半に比較的多く，トレースを実行する回数は後半の課題に多い．DESUS の指導で学習者がトレースを試みるのは課題 5 以降であり，指導なしに独自のトレースを試みるのは課題８で始めて 1 件あり，課題 10 以降の後半の課題がほとんどである．

求められた支援内容を課題別に集計したものが表 7.6 である．表 7.6 から「指定行」は課題 5 から後半の課題でほぼ全般にわたって求められていた．それに対して「最初行」の支援は，38 件のうち 25 件までが前半の課題 10 までで求められていた．「翻訳エラー」の

解説は課題 5 以前には求められていない．このことは，前半の課題で表示されたエラーが全く判別できなかった学習者が「最初行」の支援を多く求めたと考えられる．

課題 10 は，プログラムの基本構造をすべて使用する可能性があるはじめての課題である（表 7.1 参照）．学習者独自のトレースのほとんどがこの課題 10 以降である状況から判断すると，基本構造を組み合わせて構築するプログラム本来の課題で，学習者は始めてトレースが必要な本格的な行き詰まり状態に陥っていると推察できる．したがって，トレース指導を開始する時期は，プログラムの基本構造のすべてを教育した段階の課題からが適していると考えられる．

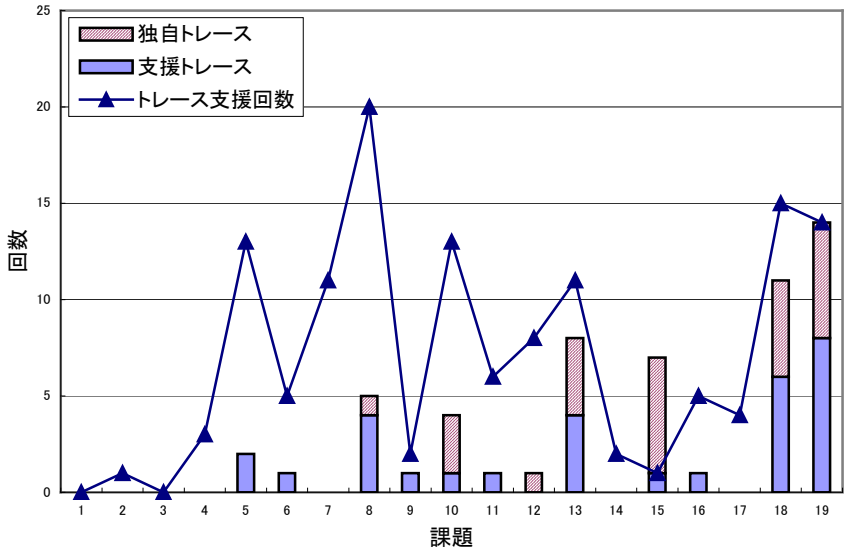


図 7.9　課題順のトレース指導回数とトレース実行回数

表 7.6　指導別課題別の支援頻度

指導内容			課題1	課題2	課題3	課題4	課題5	課題6	課題7	課題8	課題9	課題10	課題11	課題12	課題13	課題14	課題15	課題16	課題17	課題18	課題19	合計
トレース指導	位置通過	指定行変数							1			1										2
		指定行					3	1	1	1	2	5	4	3	3	2		3	2	2	8	40
		最終行				1				3										3		7
		最初行		1		1	5	4	3	8	1	2	2		2			1	2	4	2	38
		指定行通過				1				1					2					1		5
	変数推移	変数値					1			3		3	1	1	2			1		6	3	21
	条件判定														2							2
	繰り返し	for 変数					3		6	2							1			3	1	16
		for 範囲					1					2										3
エラー説明		翻訳エラー						3	1	3	1	4	3	2	6	1	1	9	2	5	1	42
命令解説		出力命令								1		2			2	1					1	7
合計			0	1	0	3	13	8	12	22	4	19	10	6	19	4	2	14	6	24	16	183

7.4　デバッグ支援システムに対する評価

7.4.1　トレース学習者の増加

DESUS のトレース指導に対する結果は，対象学習者 25 名のうち，学習者自らが学習したもの 1 名，DESUS の指導で学習した者が 8 名（32%）であった．トレース指導を口頭による指導だけで行なった 2003 年度の実習では 14%であった学習率が 32%に増加したことは，DESUS のトレース指導に一定の効果があったと考えられる．

DESUS によりトレースを実行したがトレース技術を獲得している確証が得られなかった学習者は 4 名（16%）であり，このうち 3 名はデバッグ過程で大きな行き詰まり状態に陥っていないことがトレースを独自に使用するに至らなかった理由と推定された．これらの学習者はさらに大きな行き詰まり状態が発生すれば，トレースを独自に使用した可能性がある．

また，25 名の学習者のうち 2 名を除く 23 名の学習者（92%）は，DESUS から 1 回以上のトレース指導を受けていた．これは学習者にトレースプラン生成サイクルの存在を知らせるという DESUS の目標に対する成果である．

7.4.2　トレース実行とプログラミング学習の関係

トレース実行を 3 段階に分けて分析した結果，トレースを活用した学習者ほどプログラミングの筆記試験結果が高い傾向があった．これらの学習者は，トレース技術を獲得できなかった学習者よりも多くの翻訳実行回数でプログラムを作成していた．トレースはバグを的確に発見することで無用の試行錯誤を減少させる．そのため本来はプログラム完成までの翻訳回数を減少させる働きがあると考えられるが，今回の実験ではトレースを使った学習者グループの方がデバッグ過程は長くなっていた．初期のプログラミング学習者は課題プログラムをデバッグする過程で，個別に学習したプログラミング言語やアルゴリズムをそれらが有機的に関連を持ったものとして認識する．翻訳実行回数が多いことは，この学習の機会が増加していると考えられる．つまりトレースの実行は学習者のプラン生成サイクルを増加させ，それがプログラミング学習に繋がっていると考えられる．

プログラミング教育は，一定の教育期間を要しそれを効率良く短縮することが難しい教育科目である．トレース学習が与える影響を判断するにも一定の期間が必要となる．そのためトレース学習とプログラミング学習の関係をより明確にするためには，初期段階の教育でトレース技術を獲得した学習者と獲得していない学習者の次の教育段階での学習推移

を観察し分析する必要がある．

7.4.3　トレースを学習できなかった学習者の問題

DESUS が指導したがトレースを一度も実行しなかった学習者が 10 名（40%）いた．これらの学習者がなぜトレースを実行しなかったかを調査した結果，トレース指導を受けるに相応しくない翻訳エラー状態のプログラムで指導を受けていたことが原因のひとつであった．特に指示されたトレースを実行しない割合が高かった「指定行」指導は実行時エラーメッセージを手掛かりにトレースを指示しており，そのために学習者は指示されたトレースを実行することができなかったと考えられる．このような指導の齟齬が起こった原因は，DESUS からの最初の質問への選択肢が学習者に誤解を与えた可能性が考えられる．最初の質問への選択肢を改良するとともに，翻訳エラーでトレース指導を受けることを避ける検査機能を DESUS に追加することで改善を図ることができる．

7.4.4　トレース指導以外の指導に関すること

DESUS が指導したトレースの中で学習者が最も多く実行した「最初行」指導は，システムから出されるエラーメッセージの識別ができない学習者に対する支援である．この支援が前半の課題で多く求められたことと，多くの学習者が翻訳エラーのプログラムで DESUS の指導を受けたことは，初心者のためのデバッグ支援システムには翻訳エラーに対する指導を強化する必要があることを示している．翻訳エラーは，そのほとんどがプログラミング言語に関する文法知識の不足が原因である．初期段階で起こる翻訳エラーを用いてプログラミング言語の指導を強化することで，原因が正確に掴めないまま誤りを繰り返すことが避けられる [16]．DESUS の現在の翻訳エラーに対する支援は，DESUS への親和性を高めるために付加した機能であり，効果的な支援が行なわれているとは言いがたい．この機能を強化することにより，真にトレース指導が必要な学習者に対象を絞って指導することが可能になる．

7.4.5　トレース指導の時期に関すること

学習者がトレースを実行する時期は，プログラミングの基本構造をすべて学習した後にそれらを組み合わせた課題を作成する時期と重なる．学習者にとってそれぞれの命令や制御構造を理解できても，それらの命令を複数組み合わせてプログラムを実現することには

別の困難さがある．そのため多くの学習者がこの段階で行き詰まりを感じトレース実行を試みたと考えられる．したがって，トレースが有効な行き詰まりを起こす可能性が高いこの時期に指導を開始することがトレース教育には効果的であると考えられる．

8. おわりに

本研究では，学習者のプログラミング行動を詳細に分析し，デバッグ時に学習者が用いる様々な戦術がプログラミング学習に多大な影響を与えていることを明らかにし，そのうえで，プログラミング教育の初期段階から積極的にデバッグ時に使用する戦術を教授することを試みた．戦術としては学習者自身が独自に技術を獲得することが最も少ないトレースを取り上げ，学習者の求めに応じて個別にトレースを指導するデバッグ支援システム DESUS を構築した．この支援システムを使用する環境でプログラミング教育を実践した結果，トレースを活用する学習者が大幅に増加するとともに，トレース技術を獲得した学習者はプログラミング筆記試験で高い評価を受けていた．

トレースというデバッグ戦術は，エキスパート技術者によって情報処理のあらゆる場面で使用されている．その使用目的はデバッグを効率よく進めることである．ところが初心者がデバッグで実行するトレースには，エキスパートが使用するトレースの働きの他に，プログラミング学習の新しい機会を作るという別の役割がある．人間の教師は，学習者のプログラムの誤りを直接指摘せず，学習者がその誤りを発見することを誘導するという指導方法をとることがある．そのような指導を模倣する DESUS は，トレースを指導することで学習者自身が誤りを発見する場を作るように働きかけをする．つまり DESUS のトレース指導の目的はトレースを活用して短いデバッグ過程でプログラム開発を終わらせることだけではなく，トレースという方法を使ってプログラム理解の機会を増やすことが目的である．本論の分析では，トレース学習者はトレースを使用することによりデバッグ過程が長くなるが，プログラミング学習は進んでいた．このことが，DESUS のプログラミング学習における役割を示している．

デバッグ時に使用する戦術の指導は，プログラミング教育の中心的な教育内容ではない．また，学習者自身がそれまでに経験した問題解決過程から独自に戦術を考案することもあり，指導する必要がある戦術がそれほど多くあるわけではない．しかし，トレースに関しては，学習者が独自に編み出すことは少なく，口頭で指導する教育にも限界がある．DESUS の実験から，学習者のプログラム開発環境にトレースを指導する支援機能が備えられると学習者は比較的容易にトレースを学ぶことが実証された．

DESUS は学習者のプログラムの誤りを同定してトレースを指導しているわけではない．初期の学習者にトレースを指導するには，本研究で示したような学習者の報告したプログラムの状態から推察したトレースでも一定の役割を果たす．しかし，少し進んだ学習段階

の学習者には，プログラムの誤りを正確に診断して，さらにその誤りを明らかにする最適なトレースを指導する必要がある. これを実現するには, 学習者プログラムの誤り診断と，診断に基づいた誤りに対する適切なトレースを割り出す機構が必要である．これらは高度な知的プログラミング教育支援システムを実現する分野の課題であり [5),50),51)]，DESUS の高度化を図るためには必要である.

プログラミング学習の支援を考えるには, 対象である学習者のプログラミング行動を詳細に掴む必要がある．本研究では個々の学習者が課題作成過程で翻訳したすべてのプログラムを使用してその学習者のプログラム生成過程を追跡する方法を用いた．この方法による分析は，学習者のプログラム生成過程を再現することにより詳細な行動が把握できるが，分析には非常に多くの時間を要する．本研究では詳細な分析を実施する前段階の情報収集のための自動化は行ったが部分的な自動化に留まっている．学習者のデバッグ過程の分析機能を高度化することは，プログラミング学習に関するデータ獲得が容易になり，プログラミング学習支援に関する研究に有用な道具となる.

謝辞

本論文を纏めるにあたり，有益なご教示を頂いただけでなく，辛抱強く熱心に長きにわたりご指導頂きました九州工業大学情報工学部の竹内章教授に心より感謝いたします．また，貴重なご助言を賜りました九州工業大学情報工学部の遠藤勉教授，井上勝裕教授，國近秀信准教授に深謝したします．

九州工業大学情報工学部の元教授である大槻説乎先生からは，プログラミング教育支援に関する研究の初期段階で方向性を示唆するご助言を数多く頂きました．これが本研究を遂行する大きな支えとなりました．深く感謝致します．

プログラミング教育支援の研究を共に開始した近畿大学産業理工学部の長田一興教授からは，共同研究を通して，様々なご教示とともに温かい激励を数多く頂きました．深くお礼申し上げます．同じく，プログラミング教育に共に携わり多くのご支援を頂きました大島商船高等専門学校情報工学科の岡村健史郎准教授，山口大学情報機構メディア基盤センター佐伯徹郎准教授に深謝致します．

そして何よりもこの論文を纏める機会を与えて下さいました宇部フロンティア大学短期大学部情報システム学科の元教授である吉田信夫先生に深謝致しますとともに同学科の皆様にお礼を申しあげます．

最後になりますが，三十数年にわたり私のプログラミング関係の科目を受講した学生たちに心からお礼を申しあげます．彼らはプログラミング教育にとって貴重な情報を数多く提供することによって教師としての私を育ててくれました．真剣にプログラミングを学ぼうとした学生たちと彼らが残した記録がこの研究の源泉のすべてです．

参考文献

1） H. J. Perkinson（平野知美，五十嵐敦子，中山幸夫訳）：誤りから学ぶ教育に向けて，勁草書房，(2000)

2） 上野晴樹 : 知的プログラミング支援システム INTELLITUTOR について－背景と開発思想－，情報処理学会研究会　知識工学と人工知能，37，pp.1-8，(1984)

3） 上野晴樹 : 知的プログラミング環境, 情報処理, Vol.28, No.10, pp.1280-1296, (1987)

4） W. L. Johnson, E. Soloway ： PROUST ： Knowledge-Based Program Understanding, IEEE,Vol.SE-11,No.3, pp267-275，(1985)

5） E. Wenger（岡本敏雄，溝口理一郎監訳）: 知的 CAI システム，オーム社，(1990)

6） J.C. Spohrer：MARCEL:Simulating the Novice Programmer, Ablex Publishing Corporation, (1992)

7） 藤居藤樹，渡邉豊英，田中淳志，杉江昇：PASCAL プログラム教授システムにおける誤り同定法，情報処理学会論文誌，Vol.34，No.3，pp.359-370,(1993)

8） 海尻賢二：ゴール／プランに基づく初心者プログラムの認識システム，電子情報通信学会論文誌D-II，Vol.J78-D-II，No.2，pp.321-332，(1995)

9） 伊東良二，小西達裕，伊東幸宏：プログラム問題領域上での動作説明を行うプログラミング学習支援システムの構築，人工知能学会誌，Vol.15，No.2，pp.362-375，(2000)

10） 斐品正照，徳岡健一，河村一樹：構造化チャートを用いたアルゴリズム学習支援システム，情報処理学会論文誌，Vol.45，No.10，pp.2454-2467，(2004)

11） 松田憲幸，柏原昭博，平嶋宗，豊田順一：プログラムの振舞いに基づく再帰プログラミングの教育支援, 電子情報通信学会論文誌 D-II, Vol.J80-D-II, No.1, pp.326-335, (1997)

12） 黄寧，丸山桃代，宮寺庸造，横山節雄：プログラム可視化によるプログラミング教育支援，信学技報，ET99-1，pp.1-6，(1999)

13） 山本芳人：流れ図とソースプログラムを対応させたプログラミング学習支援システムの利用，教育システム情報学会誌，Vol.20，No.4，pp.380-384，(2003)

14） 柏原昭博，久米井邦貴，梅野浩司，豊田順一：プログラム空欄補充問題の作成とその評価，人工知能学会論文誌，Vol.16，No.4 C，pp.384-391，(2001)

15） 高本明美，藤井美知子，泉直利，田中稔：誤り原因を指摘するプログラム学習支援

システムとその学習効果，教育システム情報学会誌，Vol.17，No.4，pp.533-540，(2001)

16） 江木鶴子，利光祐紀，竹内章：初心者のエラー発生要因に注目した Prolog 構文エラーへの支援，教育システム情報学会誌（投稿中）

17） 矢野将之，藤崎邦博，平嶋宗，竹内章：再帰構造の図式を導入した Prolog プログラミング学習支援システム，教育システム情報学会誌，Vol.18，No.3，pp.319-327，（2001）

18） 中村亮太，西田知博，松浦敏雄：プログラミング入門教育用学習環境 PEN，情報処理学会研究報告，2005-CE-81，pp.65-71，(2005)

19） 長慎也，甲斐宗徳，川合晶，日野孝昭，前島真一，筧捷彦：Nigari-Java 言語へも移行しやすい初学者向けプログラミング言語，情報処理学会研究報告，CE-71，pp.13-20，(2003)

20） 西輝之，劉渤江，横田一正：デバッガとの連携による C 言語学習支援システムの提案，信学技報，ET2006-136，pp.173-178，(2007)

21） 中島秀樹，高橋直久，細川宜秀：プログラミング学習のための QA サイクル - 受講者の習得度に応じた問題自動提示メカニズム，電子情報通信学会論文誌 D-1，Vol.J88-D-I，No.2，pp.439-450，(2005)

22） 知見邦彦，櫨山淳雄，宮寺庸造：失敗知識を利用したプログラミング学習環境の構築，電子情報通信学会論文誌 D-I，Vol.J88-D-I，No.1，pp.66-75，(2005)

23） 服部徳秀，石井直宏：プログラミング演習の評価サポートシステムの構築，教育システム情報学会誌，Vol.14，No.1，pp.21-28，(1997)

24） 関本理佳，海尻賢二，山形昌也：ネットワークを利用したレポート受付・評価支援システムの実現，教育システム情報学会誌，Vol.14，No.5，pp217-222，(1998)

25） 小西達裕，鈴木浩之，伊東幸宏：プログラミング教育における教師支援のためのプログラム評価機構，電子情報通信学会論文誌 D-1，Vol.J83-D-1，No.6，pp.682-692，(2000)

26） 鈴木浩之，小西達裕，伊東幸宏：抽象的データ構造を含むアルゴリズム表現に基づくプログラム評価支援システムの構築，教育システム情報学会誌，Vol.24，No.3，pp.167-185，(2007)

27） 小倉崇，藁谷大輔，櫻井孝平，古宮誠一：オブジェクト指向プログラムのためのデ

バッグ／テスト支援システムープログラムトレーサー機能についてー，信学技報，KBSE2004-36，pp.7-12，(2005)

28） J. C. Spohrer, E. Soloway and E. Pope: A Goal/Plan Analysis of Buggy Pascal Programs, Human Computer Intraction , No.1, pp.163-207, (1985)

29） W. L. Johnson ： Intention-Based Diagnosis of Novice Programming Errors, Morgan Kaufmann Publishers Inc., (1986)

30） J. C. Spohrer, E. Soloway: Analyzing the High Frequency Bugs in Novice Programs, Empirical Studies of Programmers, Ablex, pp.230-251, (1986)

31） A. Newell, H. A. Simon: Human Problem Solving, Prentice-Hall, Inc., New Jersey,(1972)

32） G. Polya（柿内賢信訳）：いかにして問題をとくか，丸善，(2005)

33） 三輪和久，杉江昇：学習の初期段階における計算機プログラミングの動的過程，人工知能学会誌，Vol.7，No.1，pp.138-148，(1992)

34） M. Matz: Towards a process model for high school algebra errors, Intelligent Tutoring Systems, Academic Press, pp25-50, (1982)

35） E. Soloway, J. Bonar, K. Ehrlich: Cognitive Strategies and Looping Constructs: An Empirical Study, Comm. of the ACM, Vol.26, No.11, pp.853-860, (1983)

36） E. Soloway, K. Ehrlich: Empirical Studies of Programming Knowledge, IEEE Trans. on Software Eng., Vol.SE-10, No.5, pp.595-609, (1984)

37） 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，Vol.6，No.3，pp.26-36，（1989）

38） 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 プログラムの誤り分析，CAI 学会誌，Vol.7，No.2，pp.80-91，（1990）

39） 江木鶴子，岡村健史郎，長田一興：COBOL プログラムの誤りに関するデータベースとプログラミング教育への利用，CAI 学会誌，Vol.9，No.2，pp.51-62，(1992)

40） 江木鶴子，有銘興建，長田一興：プロトコル分析による学生プログラマのプログラム生成過程分析，CAI 学会誌，Vol.11，No.4，pp.195-206，(1995)

41） Tsuruko Egi and Kazuoki Osada : A Descriptive Model of Student Program Generation Based on a Protocol Analysis , Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, The 9th International

Conference Fukuoka, p.784, (1996).

42) 江木鶴子，前田直樹，長田一興：戦術プランを用いる学生プログラマのプログラム生成過程モデル，電子情報通信学会論文誌 A，Vol.J79-A，No.10，pp.1742-1753，(1996)

43) 海保博之，原田悦子：プロトコル分析入門，新曜社，(1993)

44) M.W. van Someren, Y. F. Barnard, J. A. C. Sandberg: The Think Aroud Method, Academic Press, (1994)

45) 加藤隆：計算機ユーザの認知行動原理を探るための一手法，情報処理，第 26 巻，弟 9 号，pp.1106-1109，(1985)

46) Tsuruko Egi and Akira Takeuchi: An Analysis on a Learning Support System for Tracing in Beginner's Debugging, Supporting Learning Flow through Integrative Technologies, IOS Press, pp.509-516, (2007).

47) 江木鶴子，竹内章：プログラミング初心者にトレースを指導するデバッグ支援システムの開発と評価，日本教育工学会論文誌，Vol.32，No.4，pp.369-381，(2009)

48) H. Mandl, A. Lesgold 編（菅井勝雄，野嶋栄一郎監訳）：知的教育システムと学習，共立出版，(1992)

49) 教育システム情報学会編：教育システム情報ハンドブック，実教出版，(2001)

50) D. Sleeman，J. S. Brown（山本米雄，岡本敏雄監訳）：人工知能と知的 CAI システム，講談社，(1987)

51) 大槻説乎，山本米雄：知的 CAI のパラダイムと実現環境，情報処理，Vol.29，No.11，pp.1255-1265，(1988)

資料

1. 2007 年度後期 Java 入門試験問題

Java 入門　2007 年度後期試験　(学籍番号　　名前　　)

［1］ 下記のプログラムに関して、次の質問に答えて下さい。

1-1　このプログラムで宣言されているクラス名を答えて下さい。

1-2　このプログラムで宣言されているメソッド名を答えて下さい。

1-3　System.out.println( ) と　System.out.print( )の命令の違いを説明して下さい。

1-4　このプログラムは、実行すると何を出力しますか。その内容を記述して下さい。

```
/*
プログラム名：DrawGraph.java
*/

public class DrawGraph{
    public static void main(String[] arges){
        for(int i=0;i<10;i++){
            printGraph(i);
        }
    }
    public static void printGraph(int x){
        for(int j=0;j<x;j++){
            System.out.print("*");
        }
        System.out.println("");
    }
}
```

［2］ 下記のプログラムに関して、次の質問に答えて下さい。

２-１　このプログラムを実行すると、A の命令は何回実行されますか？

２-２　このプログラムを実行すると、B の命令は何回実行されますか？

２-３　B の命令が実行された直後、i の値はどのような値になっていますか？

```
/*
プログラム名：Count.java
*/

public class Count{
    public static void main(String[] arges){
        for(int i=0; i<3; i++){
            System.out.println(i);   ←---------- A
        }
        System.out.println("end");   ←---------- B
    }
}
```

［3］ 下記のプログラムについて、以下の質問に答えて下さい。

３-１　このプログラムで使われている**nen%4==0** の条件の意味を日本語で答えて下さい。

３-２　このプログラムで使われている**nen%100!=0**の条件の意味を日本語で答えて下さい。

３-３　条件**nen%4==0　&&　nen%100!=0**の意味を日本語で答えて下さい。

３-４　このプログラムに、2000 を入力すると、どのような出力がされるかを答えて下さい。

３-５　このプログラムに 1998 を入力すると、どのような出力がされるかを答えて下さい。

```
/*
 プログラム名:Uruu.java
*/
import java.io.*;

public class Uruu{
    public static void main(String[] arges)throws IOException{
        String nen_in;
        int nen;

 BufferedReader kbd=new BufferedReader (new InputStreamReader(System.in));

        while(true){
            System.out.print("年を西暦で入力してください。");
            nen_in=kbd.readLine();
            try{
                    nen=Integer.parseInt(nen_in);
                    if(nen%4 == 0 && nen%100 !=0){
                    System.out.println("閏年です");
                    }
                    else if (nen%400 == 0){
                            System.out.println("閏年です");
                    }
                    else{
                            System.out.println("閏年ではありません");
                    }
            }
            catch(NumberFormatException e){
                     System.out.println("入力が間違っています。");
                     continue;                }
            break;
        }
    }
}
```

［4］ 下記のプログラムは、入力された数値のうち最大値と最小値を出力するものです。このプログラムに関して、次の質問に答えて下さい。

4-1　入力の値として、８　３　４　６　．　がこの順に入力された場合、このプログラムはどのような出力をしますか？　出力される通りに記述して下さい。このプログラムでは無駄に思える出力もされていますので注意して下さい。

4-2　このプログラムには誤りがあります。したがって、前問題の出力結果は、正しい結果ではありません。このプログラムの誤りを見つけてその理由と、それをどのように訂正したらよいかを答えて下さい。

```
/*
プログラム名：MaxMin.java
*/

import java.io.*;
public class MaxMin{
    public static void main(String[] arges) throws IOException {
        String   b ;
        int dt, max, min;

        BufferedReader a = new BufferedReader(new InputStreamReader(System.in));
        max=-100000;
        min=-1;

        System.out.println("pass1:"+min);

        dt=-1;
        System.out.println("Type in suu.");
        System.out.println("End mark is .");
        b =a.readLine();
        while(!(b .equals("."))){
                try{dt=Integer.parseInt(b);
                    if(dt>max){
                            max=dt;
                        }
                    else if(dt<min){
                            min=dt;
                            System.out.println("pass2:"+min);
                        }
                    b=a.readLine();
                    }
                    catch(NumberFormatException e){
                            System.out.println("Type in suu.");
                    b=a.readLine();
                    }
        }
        System.out.println("Max="+max);
        System.out.println("Min="+min);
        System.out.println("pass3:"+min);
    }
}
```

［5］ 下記のプログラムは、配列に格納された数値を分類する（ソート）プログラムです。このプログラムは泡立ち法というアルゴリズムで作成されています。このプログラムに関して、次の質問に答えて下さい。

5-1　このプログラムを実行した際、＊の時点で変数 size には、どのような値が入っていますか。

5-2　同じく、＊の時点で変数 kazu[2]には、どのような値が入っていますか。

5-3　＊＊時点で、kazu[2]には、どのような値が入っていますか。

5-4　＊＊＊時点で、kazu[2]には、どのような値が入っていますか。

5-5　もし配列 kazu の値が、{2, 4, 5, 5, 18, 34} であったなら、＊＊＊＊の命令は、何回実行されますか。

```
/*
プログラム名：Sort.java
*/

public class Sort{
    public static void main(String[] arges){
     int x, temp, size;
     int[] kazu = {11, 32, 8, 22, 6};
     int swaped = 0;

     size=kazu.length;
                                  ←---------- *
          while(swaped == 0){
                swaped = 1;              ←---------- ****
                for(x=0; x<size-1; x++){
                     if(kazu[x] > kazu[x+1]){
                          swaped = 0;
                          temp = kazu[x];
                          kazu[x] = kazu[x+1];
                          kazu[x+1] = temp;
                     }
                }
          }
                                           ←---------- **
          for(x=0; x<size; x++)
                System.out.println(kazu[x]);
    }                                      ←---------- ***
}
```