Vol.2010-SE-167 No.11
2010/3/18

ブロック構造の可視化によるプログラミング学習支援環境 azur 〜関数の動作の可視化〜

今泉俊幸[†]　橋浦弘明[†]　松浦佐江子[†]　古宮誠一[†]

プログラミングを学習する際には，記述されたプログラムの内容を正しく理解することが必要である．しかし，初学者にとってプログラムの実行過程を理解することは難しい．その原因として，初学者は制御構造，関数を含んだプログラムの適切な動作イメージを作成できていないということに着目した．本研究では，プログラミング初学者に対し，制御構造と関数の動作を理解することを支援するツールを開発した．本稿ではそのシステムの内容を明らかにするとともに，ツールの有効性を確認している．

Azur,Program Learning Environment through Visualizing Block Structure. 〜Apply to Function〜

Toshiyuki Imaizumi[†]　and Hiroaki Hashiura[†]　and Saeko Matsuura[†]　and Seiichi Komiya[†]

When a student learns programming, it is necessary to understand the contents of described program definitely. However, it is difficult to understand the practice process of the program for a student. I paid my attention to that the student cannot make the appropriate movement image of the program including the control structure and function as the cause. In this study, I developed the tool which supported that a student understood movement of the control structure and function. I clarify the content of the system by this report and confirm the effectiveness of the tool.

1. はじめに

ソフトウェア開発に関わる技術は日々進歩している．開発者は新たな技術，開発手法への迅速な対応が必要である．そのため，開発者の教育にかかるコストや時間は増大してきているが, 複雑化, 大規模化するソフトウェアの開発業務が多忙なことから，開発者に対し十分な教育時間を確保することは難しい．そのため，企業では大学などの教育機関に対し，ソフトウェア開発に必要な基本的な技術を身につけさせることを望んでいる．

ソフトウェア開発を行う為には，プログラミング技術の習得が必要不可欠である．プログラムは期待するコンピュータの動作をプログラミング言語で記述したものである．しかし，その静的な構造と動的な構造は異なるため，プログラミング初学者が，プログラムの記述とその動作内容の対応を正しく理解することは困難である．そのため，教授者は問題を抱えた学習者に対し，個別に対応する必要がある．しかし，一般に教授者に対し，学習者の人数が多いことから，全ての学生に対応しきれていないという問題がある．プログラミングの動作内容を理解できない学習者が独学で学習することは困難であるため，プログラムを正しく理解する能力が欠けたまま進級し，以後の授業についていけなくなってしまう学習者が生じている．

そこで，本研究ではプログラミング初学者に対し，教授者が隣にいるかのようにプログラムの動作過程の理解を支援する開発環境を提案する．対象とする言語は C とした．

本稿では，本ツールの機能を明らかにし，このツールが学習者のプログラムの理解に与える効果を示す．

2. プログラミング教育の現状

教育機関でのプログラミング教育は，一般にプログラミング言語の文法の学習，基本的なアルゴリズムとその実装方法の学習，アプリケーション開発といった順序で行われる．

このような順序で教育を行う場合，先行する学習項目を習得しなければ，次の学習項目を習得することができない．そのため，最初の項目でつまずき，理解に失敗した学習者は，以後の授業の内容をほとんど理解することができなくなってしまう．

しかし，実際には，多くの学習者は文法を学習した後でも，十数行程度の簡単なプログラムさえも記述できていないというのが現状である．

多くの学習者が簡単なプログラムさえも理解できない理由の一つとして，記述されたプログラムの動作を正しく理解できていないということがあげられる．

[†] 芝浦工業大学大学院
Graduate School of Engineering, Shibaura Institute of Technology

ⓒ2010 Information Processing Society of Japan

高級言語における変数や関数呼び出しといった動作を完全に理解するためには，コンピュータのメモリの利用方法，プログラムカウンタ，二進数，十六進数，スタックなどといった低レイヤの知識が必要となる．しかしながら，プログラミング初学者の多くはそういった原理を十分理解していないことが多い．また，学習用のプログラムは一部を除いて途中経過を十分出力するようになっていない．

このような状況下で，学習者は，自分が作成したプログラムとプログラムの最終的な実行結果をもとに，自分でプログラムの最終結果が導き出される過程(動作のモデル)を推測しなければならない．しかし，学習者の多くは，命令がどのような順番で実行され，変数の値がどう変化するかという，プログラムの実行過程を深く考えずに試行錯誤を繰り返すことでプログラムを作成している．

そのため，学習者の多くはプログラミングの初歩の段階で，記述された静的構造が表すプログラムの動作のモデルの推測を誤る．これまでの学習方法ではそのようなモデルを修正する機会が十分与えられないため，このような誤りが蓄積されていくと，小さなプログラムであっても，プログラムがどのように実行されるのかを理解することができなくなる．

このような誤りを修正するためには，プログラムの一文一文がどのような動作に対応しているのかを詳細に示す必要がある．

一方，プログラムの詳細を必要以上に示すと，説明が冗長になり，かえって学習者の理解の妨げになることがあるため，詳細な説明は学習者が誤った理解をしている部分に絞って行う必要がある．

通常，学習者の誤りはそれぞれの学習者ごとに異なっている．このため，教授者は学習者に対し，個別に対応する必要があるが，学習者と教授者の人数比を考えれば，時間的な制約から全ての学習者に適切な対応を行うのは不可能である．

さらに，一度説明しただけでは学習者に動作のモデルが定着するとは限らないため，反復的なサポートが必要とされていることも軽視できない．

3. プログラムの作成過程とつまづきの原因

プログラムの動作過程を理解するためには，変数のスコープ，メモリイメージ，配列やポインタといったデータに関する知識と，選択，反復構文，関数といったプロラムの制御フローについての知識を理解する必要がある．本研究ではまず，プログラミング初学者に対し，制御フローの理解をさせることに注力する．初学者であってもデータは print 文を用いて画面に出力することが可能であるが，制御フローを確認することは難しいため，優先的に対処する必要があると考えた．

一般に，プログラムを作成する過程は以下の図のようなサイクルとなっている．

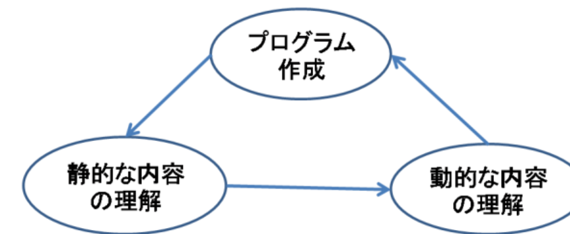

図 1　プログラムの作成過程

まず，学習者はソースコードからプログラムの静的な側面を理解する．次に，プログラムを実行したときにどのような動作が行われ，出力が行われるのかなどの動作内容を考え，理解する．そして，プログラムの目的と異なる動作を行う部分について修正を行う．これを正しい出力結果が求められるまで繰り返す．

このように，プログラムを理解するにあたっては，静的な側面と動的な側面の二つの理解が必要となる．

上に述べた二つの観点から，制御フローにおいて，多くの学習者が正しく理解できていないものを考える．静的な側面の理解としては，選択構文の結果，どの部分が実行されるのかということや，反復構文がどの範囲を反復するのかといったような，正しい実行範囲を理解していないというものがある．特に，制御構文がネストした際に実行範囲の認識を誤ることが多い．

動的な側面の理解としては，どのような実行経路を通るのかを正しく理解できていないこと，関数においてデータがどう受け渡されているのか理解できていないということがある．

著者らは学習者が制御フローを正しく理解できない原因として，以下の二つがあると考えている．

一つ目は，制御構文，関数の動作に対し，適切なイメージを作成することができていないということである．長谷川ら 1)は，学習者に対し，制御構造を含んだプログラムの動作の理解と，制御構造に対する動作イメージの有無との関係を調査している．その結果，制御構造に対し何らかの具体的な動作イメージを持っている学習者は，イメージを持っていない学習者よりも，制御構造の動作を正しく記述できるとの調査結果を示している．つまり，制御構造を理解するためには，学習者に対し何らかの動作イメージを提示することが重要と考えられる．

ここで，どのような動作イメージが適切かを考える．プログラミングの熟練者は制御構造がネストした場合，ネストが深いという表現を用いることが多い．そのため，制御構造に対し，深さを表現できる階層的なモデルを構築することが，制御構

ⓒ2010 Information Processing Society of Japan

造の理解に有効だと考えられる．

長谷川ら 2)は，学習者が制御構造に対しどのようなイメージを持っているかについても調査を行っている．調査の結果，反復構文に対し，階段型や数え上げ型といった，深さの表現が困難なイメージを持っている学習者が 3 割，深さの表現が可能なループ型のイメージを持っている学習者が 7 割存在したと報告している．

深さの表現が困難なイメージを持っている学習者は，反復構造がネストした場合にうまく動作イメージを作成できなく，正しいプログラムの動作を推測できないということが考えられる．そのため，制御構造の含んだプログラムを理解させるためには，学習者に制御構造のネストに対応できる，適切なイメージを習得させることが必要である．

二つ目は，学習用のプログラムの問題である．2 で述べたように学習用のプログラムは一部を除いて途中経過を十分出力するようになっていないことである．

これらの問題点を解決するため, 著者らは様々なプログラムやその入力値に対し，プログラムの構造と実際にプログラムの実行過程をリアルタイムで示す環境が必要であると考えた．

4. 関連研究

プログラムの動的な側面の理解を支援するツールとして，シンボリックデバッガとプログラムトレーサーが存在する．どちらのツールもプログラムの実行過程を示すことが可能である．

シンボリックデバッガは，プログラムを最初から最後まで一気に実行するのではなく，ソースコードの行単位で実行を制御することができる．また，任意の時点で変数の値を取得することができる．そのため，ユーザはプログラムの実行経路を認識することが可能である．また，条件式で参照されている変数の値を追うことで，なぜそのような実行経路を通ったのかを推測することも可能である．

プログラムトレーサーはプログラムが実行した命令の順に，命令の内容やその出力結果，変数の値などの内部状態などを出力することができる．トレーサーはプログラムの実行過程を一気に見ることが可能であるため，ユーザはプログラムの全体の流れを見ることができる．

しかし，どちらのツールも，プログラムの実行過程がなぜそうなったのかという事について，制御構文の意味と対応づけて示すことはしていない．そのため，プログラムの制御構造を表す構文の動作モデルを利用者が持つことに寄与しない．

プログラムの静的な側面の理解を支援するツールとして，ソースコードを視覚的に表示するプログラムビジュアライザと呼ばれるツールが存在する．

プログラムの制御構造に着目したビジュアライザとして，喜多ら 3)は Avis を開発している．Avis は学習者が記述したプログラムから，プログラムのフローチャートと，取りうる実行経路の一覧をフローチャートとして出力する．Avis のように，プログラムからフローチャートやそれに類似するものを作成するツールは数多く存在する．しかし，フローチャートはプログラムの階層構造を示すことには適しておらず，ネストに対処することが可能な，適切な動作イメージを示すことができない．

新開ら 4)はプログラムのアルゴリズムを逐次，選択，反復の制御構造を用いて，擬似言語で記述し，擬似言語で記述されたプログラムのステップ実行を行うことができるツールを開発している．ツール上で制御構造は階層的に表示されるため，制御構文がネストした際の実行範囲を明確に示すことができている．また，擬似言語で作成したコードを C 言語に変換することができるため，実際のプログラムとの対応も見ることができる．しかし，擬似言語上では選択処理を if 文，反復処理を while 文でしか表現できず，switch，do-while，for 文などの動作を身につけることができない．

関数の動作の理解を目的としたツールとして，寺田ら 5)は ETV を開発している．ETV はトレース情報を用いたデバッガとなっている．そのため，通常のデバッガの機能に加え，逆戻り実行が可能となっている．ETV はソースコードの行単位の実行を行い，次に実行される行をハイライトする．プログラムを実行している際に関数呼び出しが行われた際，現在のソースコード表示画面の横に，新たにソースコード表示画面を作成する．関数呼び出しを行ったソースコード画面を残したまま，新しく作成した画面で実行行の表示を行うようにする．この機能を用いることで，ユーザは関数呼び出しの経路を一覧することができる．そのため，関数の呼び出しが終了したあと，どの行に戻ってくるかと言うことを理解することができる．しかし，ETV は変数の変化については，ステップ毎の変数の値を表示するだけにとどめており，関数の引数と返り値の扱いや，スコープといったものを理解させることができていない．

松田ら 6)は再帰関数の動作の理解に注力したツールを開発している．このツールではユーザにプログラムの部品を組み合わせることで再帰関数を表現させている．ツールは作成したプログラムにおいて，変数の値がどのように変化するのかをアニメーションで表現する．変数がどのように変化していくかという履歴を一覧することができるため，再帰呼び出し毎に変数の値がどのように変わっていくのかを理解させることが可能である．しかし，部品を組み合わせてプログラムを作成するという性質上，作成されるプログラムには制約がかかってしまう．

以上で述べたように，擬似言語ではなく実際の言語で記述されたプログラムに対し，様々な制御構文と，制御構造がネストした際の動作のモデルを学習者に作成させるツールは存在していない．また，関数の実行過程を示しながら，引数，返り値，スコープについて学習者に理解を促すツールは存在していない．

ⓒ2010 Information Processing Society of Japan

5. 提案するプログラム理解手法

本研究では，学習者に制御構造と関数についての動作の正しいモデルを作成させる事を目的として，教育用の統合開発環境を提案する．ツールが持つべき機能として，以下の機能を考える．

- プログラムのブロック構造の可視化
- 制御構文ごとの動作の違いの可視化
- 関数呼び出しの際に行われている動作の可視化
- プログラムのステップ実行
- ブロック構造上でのプログラムの実行過程の可視化

まず，プログラムの静的な構造を理解させるために，プログラムのブロック構造の表示を行う．ブロック構造を示すことで，制御構造がネストした時でも，それぞれの実行範囲を正しく認識することができる．このブロック構造は，ネストに対応する動作イメージの構築を支援するために，階層的な形で表示を行う．

また，制御構文には，反復や選択といったある機能を実現するために，複数の種類の構文が存在している．それらの動作のモデルを作成させ,違いを認識させるために，それぞれの構文毎に，適切な可視化を行う必要がある．

次に関数についての支援として，関数呼び出しの際に行われる動作の表示を考える．関数呼び出しの際には，実引数が仮引数にコピーされ，関数内で宣言された変数などがスタックに積まれる．そして，return 文が実行されると，スタックで確保していた領域を解放し，実行元に返り値を返す．通常のデバッガでは示されない，この詳細な動作内容を可視化することで，関数の動作のモデルを作成できると考えられる．

次にプログラムのステップ実行機能があるが，これはプログラムの実行過程を認識するために必要である．

しかし，プログラムのブロック構造の表示とステップ実行機能だけでは，動作のモデルを構築できない．プログラムの動作が，ブロック構造上のどことどう対応しているのかを表示することで初めて動作イメージを構築できると考える．そのために，ステップ実行によって取得した，プログラムの実行位置が，ブロック構造上のどこに対応するのかを表示する機能が必要となる．

また，ソースコード，ソースコードの静的な側面，プログラムの動的な側面の三つを対応づけて理解するためには，それらの間でのフィードバックが容易でなければならない．そのため，これらの過程を連続して行うことが，学習者の理解にとって効果的であると考えられる．そこで，先にあげた機能を別々のツールとして提供するのではなく，一つの開発環境としてまとめて実行できるようにする．また，学習者が利用方法の習得に手間どらないように，必要な情報だけを表示する簡素なユーザインターフェースを提供する．

6. プログラム理解支援環境

6.1 ツールの機能

本ツールの機能はソースコードエディタ，プログラム可視化，逐次実行の三つから構成される．それぞれの機能について以下で順に説明する．

6.1.1 ソースコードエディタ

ソースコード記述時に一定時間が経過すると，記述されたソースコードの構文チェックを行い，コンパイルエラーとなる行をハイライトする．短い時間でエラーチェックを行うことにより，複数のエラーが絡み合い，大量のエラーが検出され，どう直したらよいのかわからなくなるといった事態が少なくなることが期待させる．学習者はエラーメッセージに注目しないことが多く，どこが間違いなのか発見することができずに諦めてしまうことが多々あるため，エラー行のハイライトはモチベーションを保つ為にも効果的である．

6.1.2 可視化

(ア) ブロック構造の可視化

プログラムのブロック構造を図 2 のように階層的に表示する．ソースコードと，可視化したブロック構造を同じ高さに揃えることで，それぞれの対応がわかりやすいようにしている．

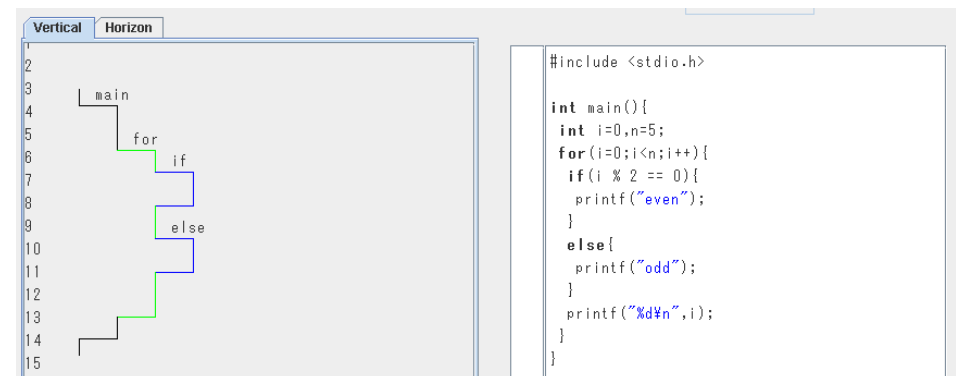

図 2 ブロック構造の可視化

関数のブロックを黒，実行中の関数を赤，反復構文のブロックを緑，選択構文のブロックを青色の線で表示している．switch 文の case，default 節は文法上ブロックとみなされないが実行範囲を明確にするために，他のブロックと同じように

ⓒ2010 Information Processing Society of Japan

表示している. また, break 文がなく fall through が発生するブロックに対しては, 破線で表現することで, fall through が起こることを示している.

また, 再帰関数のように, ある関数が複数回呼び出された時, ブロック構造を複製して順次右側に表示していくことで, スタックに積まれて行く様子を見せている.

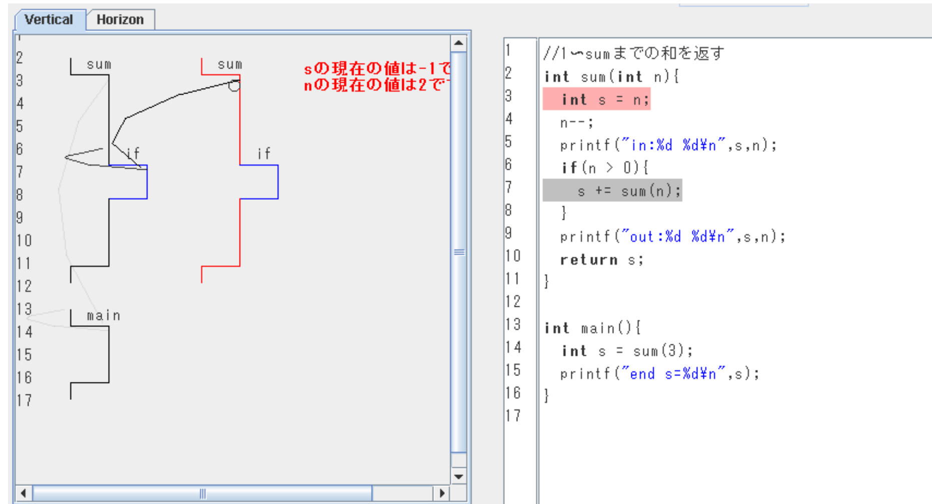

図 3　再帰関数の可視化

(イ)　プログラム実行の軌跡の表示

ステップ実行を行う際に, プログラムの実行行が, 可視化された構造とどう対応付いているかを示す為に, ボールをプログラムカウンタと見立て, 構造上を動くようにした. このボールの実行の軌跡を表示し, 残して置くことで, 反復構文の動作と, 関数呼び出しからの復帰場所についての理解を支援することができる.

6.1.3 逐次実行

(ア)　ステップ実行

プログラムの実行制御として, 行毎のステップ実行を行うことができる. 学習者にすべての実行過程を見せるために, ステップインのみをサポートしている.

(イ)　変数の値の表示

ステップ実行時には, 次に実行する行で参照している変数の値と, 直前のステップ実行で変化した変数の値を表示している(図 4). これにより, 注目する部分を少なく, 明確にすることができる.

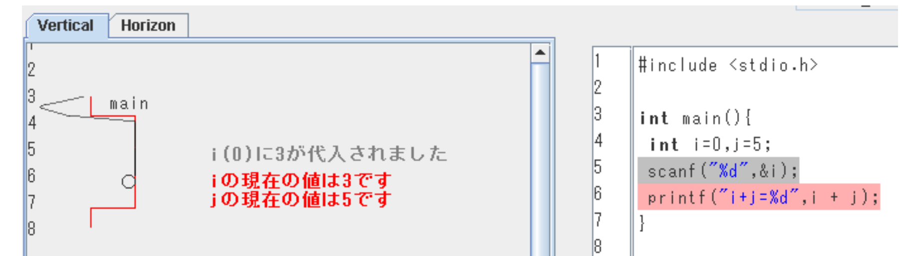

図 4　変数の変化と参照する変数の表示

(ウ)　関数の返り値の表示

関数に返り値があった場合, ボールに色をつけ, 返り値があることを示し, その値を表示する. これにより関数の返り値がどの変数に代入されるのか, 返り値が使われていないのかということを知ることができる.

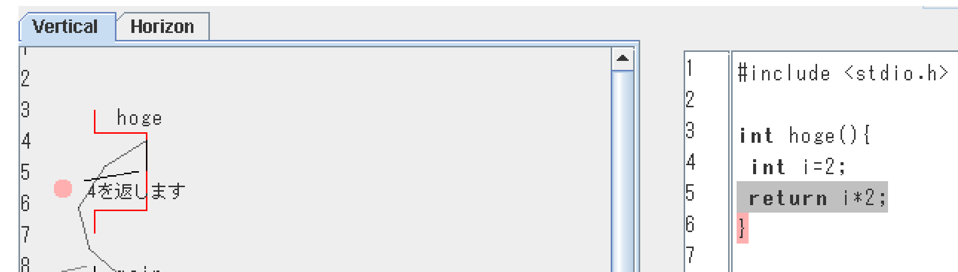

図 5　関数の返り値の表示

6.2 ツールの構成

本ツールでは, gcc と gdb を用いることで, プログラムの実行を制御している. システム構成図は以下のようになる.

ⓒ2010 Information Processing Society of Japan

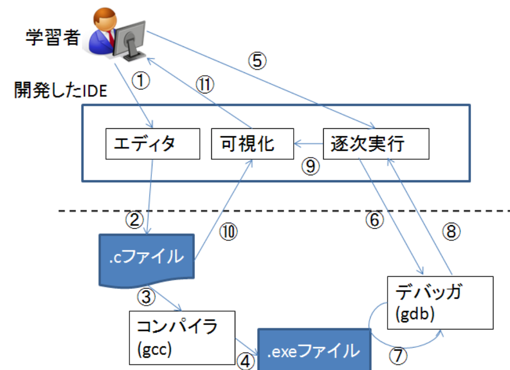

図 6 ツールの構成図

7. 評価実験

7.1 実験目的と概要

本ツールを用いてプログラムを実行することにより，プログラムの構文で表されたプログラムの実行過程を正しく認識できるようになるかを調査するために，実験を行った．今回調査を行ったのは，制御構文がネストした時の実行範囲，for 文がネストした時の動作，再帰関数の動作過程の理解である．

- 被験者

  芝浦工業大学 電子情報システム学科一年生 8 名，二年生 17 名，情報工学科三年生 6 名

- 実験手法

  初めに被験者に対し実力テストを行い，正しく実行過程を認識しているかどうかを確認した．その後，全 26 問の課題を渡し，二週間の作業期間を設け，期間中にツールを用いて課題を解かせた．課題を解いた後再度実力テストを行い，実行過程を正しく認識できるようになったかを確認した．

  実力テストとして，if 文のネスト，for 文のネスト，再帰関数をそれぞれ含む，三問を出題した．被験者には図 7 のように，配布したプログラムに対し，実行行の変化とその時の変数の値，プログラムの出力を記述させた．最後に行った実力テストは，最初に出題したプログラムの変数の値を少し変更しただけの，ほとんど同じプログラムを出題した．最後にアンケートを行い，今まで理解できなかった部分が理解できるようになったかどうかということと，ツールについてのコメントを記述させた．

配布した課題はデバッグ，穴埋め，スクラッチの三種類に分かれており，問題数は表 1 の通りである．

```
1   int main(){
2       int i=-1,j=0;
3       int n=0;
4       scanf("%d",&n);
5       for(i=0;i<n;i++){
6           j = i*2;
7           printf("%d ",j);
8       }
9       printf("¥n");
10  }
```

実行行	出力	i	j	n
1				
2		-1	0	
3				0
4				2
5		0		
6			0	
7	0			
5		1		
6			2	
7	2			
5		2		
9	¥n			

（変数の値：i, j, n）

図 7 実力テストの解答例

表 1 出題した問題の種類と個数

問題が含む要素	デバッグ	穴埋め	スクラッチ
制御構造のネスト	3	6	6
if 文と else if 文	2	1	1
for 文のネスト	1	1	1
再帰関数	1	1	1

7.2 実験結果と考察

実力テストの結果を分析し，それぞれの問題毎に誤り箇所を分類した．if 文のネストの問題の誤りとして，構文のブロック範囲の認識の誤りと，連続する if 文を else-if 文のように扱っている誤りが存在した．for 文の誤りとしては，構文のブロック範囲の誤り，内側の for 文の初期化処理が一度目しか実行されず，二回目以降のループでループカウンタの値が正しく初期化されていない誤り，支離滅裂な動きをする誤りの三つが存在した．再帰関数の誤りとしては，再帰呼び出しの呼び出し先の行が分からない誤り，再帰呼び出しされた関数の return 文を実行した後，どこに実行行が戻るか分からない誤り，再帰呼び出しされた関数ごとに変数を区別していない誤りの三つがあった．それぞれの誤りについて，実験前と実験後の実力テストの結果を表 2 にまとめた．

ⓒ2010 Information Processing Society of Japan

表 2　実力テストの結果

誤り数	if 文		for 文			再帰		
	範囲	if 文	範囲	初期化	その他	呼び出し	復帰	変数
実験前	3	2	2	6	3	8	2	5
実験後	0	0	1	2	2	6	1	1

誤りが改善された被験者とそうでない被験者の違いを考察するために，ツールの利用ログを分析した．それぞれの誤り内容の部分を，何回ステップ実行で確認しているかの回数を計測したものが表 3 である．

表 3　要素毎のステップ実行の回数

要素	誤りが修正された被験者の実行回数	誤りが修正されなかった被験者の実行回数
構文のブロック範囲	5, 11, 18, 26	0
if と else if 文	14, 29	−
for 文の初期化	1, 4, 5, 6	2, 3
支離滅裂な for 文	9	1, 3
再帰呼び出し	5, 7	3, 4, 6, 9, 9, 10
再帰からの復帰	15	2
関数毎の変数	1, 7, 8, 9	3

表 3 から，誤りが修正されなかった学生の多くは，修正された学生に比べステップ実行の回数が少ないということが分かる．そのため，これらの被験者も，より多くの回数ツールを用いることで，該当する要素の実行過程を正しく認識できるようになると考えられる．しかし，再帰呼び出しについては，多くの回数ステップ実行を行っているにも関わらず，理解できていない被験者が多い．この原因を探るために，アンケートの内容を分析した．

アンケートを分析した結果，再帰呼び出しを理解できなかった 6 人中 5 人は実験前には関数の動作過程すら理解していなかったことが分かった．これらの被験者については，再帰関数の前に，単純な関数を利用するプログラムの実行過程をツールで確認することで，対処が可能であったのではないかと推測する．

残りの 1 人の被験者に関しては，ステップ実行での確認回数が 3 回であるため，回数不足であることが理解できなかったことの原因と考えられる．

以上の結果から，本ツールを用いて，ある程度の回数プログラムの実行過程を確認することで，プログラムの実行過程を理解できるようになると考えられる．

8. まとめと今後の課題

本稿では，プログラミング教育において，制御フローの理解が難しいという問題点に着目した．制御フローを理解するためには，様々な制御構文の動作と，制御構文がネストした場合の動作，関数の呼び出しと戻りの動作について，正しい動作のモデルを作成する必要がある．そこで本研究では，プログラムの構造を可視化し，プログラムの動作過程とプログラム構造を対応させて示すツールを開発した．そして，ツールの評価実験を行うことで，ある程度の回数ツールを仕様することで，学習者が制御フローを正しく理解できるようになることを確認した．

今後の課題として，まだ実装ができていない，各制御構文ごとのブロック構造の表示方法の変更，関数の引数，返り値，変数のスコープについての可視化を進めることがある．また，反復構文中での break 文，continue 文については現在可視化を行っていないため，対応することを考えている．

参考文献

1) 長谷川聡，山住富也:プログラミング教育における制御構造理解のためのイメージ形成の効果，情報処理学会全国大会論文集, Vol.51, No.1, pp.295 -296 (1995).
2) 長谷川聡，山住富也: プログラミング教育と学習者のイメージ形成(その 2)，名古屋文理短期大学紀要, Vol.23 , pp.9 -14 (1998).
3) 喜多義弘,川添貴議,片山徹郎: Java プログラム自動可視化ツール Avis におけるクラス構造可視化のための拡張，電子情報通信学会技術研究報告,Vol.105,No.490,pp.7-12(2005).
4) 新開純子，炭谷真也: プロセスを重視したプログラミング教育支援システムの開発，日本教育工学会論文誌, Vol.31, pp.45 - 48 (2008).
5) Minoru,T.: ETV: a program trace player for students, Proc. ITiCSE '05, Vol.37, No.3, pp.118-122(2005).
6) 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,Vol.80,No.1,pp.326-335(1997).

ⓒ2010 Information Processing Society of Japan