Vol.2014-SE-183 No.5
2014/3/19

プログラミング学習支援環境 AZUR —ブロック構造と関数動作の可視化による支援—

古宮誠一[†1]　今泉俊幸[†2]　橋浦弘明[†3]　松浦佐江子[†4]

プログラミングの初学者にとって，プログラムの記述から，そのプログラムの挙動を思い描くことは難しい。その原因は，初学者がプログラムの実行を制御する構文の記述内容が変わるとプログラムの挙動がどのように変化するかということを理解できていないからだと考えられる。そこで著者らは，制御構文と再帰関数を採り上げ，これらの構文の記述内容によってプログラムの挙動がどのようになるか，プログラムの挙動を可視化するシステム AZUR を開発した。そして，AZUR をプログラミングの演習授業に導入し，学生達が使用した結果を分析することにより，AZUR の可視化機能がプログラミングの学習に有効であることを示している。

A Programming Learning-Support Environment “AZUR”: Support by Visualizing Block Structure and Program Function Behavior

Seiichi KOMIYA[†1]　Toshiyuki IMAIZUMI[†2]
Hiroaki HASHIURA[†3]　Saeko MATSUURA[†4]

It is difficult for beginner programmers imagine how a program does behave. It is thought that the cause is because beginners cannot understand it how the behavior of the program changes when control statements (e.g. if-statement, while-statement, switch-statements) are changed. Therefore they focused on control statements and recursive function, and developed system AZUR which visualized the behavior of the program. And they introduce AZUR into a programming class and verified that a visualization function of AZUR is effective for learning of the programming by analyzing the result that students used.

1. はじめに

ソフトウェアを開発するには，プログラミング技術の習得が必要不可欠である。プログラミングができるようになるには，プログラムの記述からそのプログラムの挙動を正しく思い描くことができるようになる必要がある。しかし，プログラミングの初学者は，プログラムが期待どおりの処理結果を出力するかどうかについては注目しているが，プログラムの挙動にはほとんど注意を払っていない。そのため，プログラミングの初学者が想像するプログラムの挙動と，プログラムの実際の挙動とが一致しないことが少なくない。そのような認識のずれが集積すると，プログラムがどのように動作するか理解できなくなるため，教授者は初学者の認識にどのような誤りがあるのかを推測し，適切なアドバイスを与える必要がある。しかし，一般に，一人の教員が指導しなければならない学習者の人数が多いので，個人指導のような手厚さで，一人の教員がすべての学習者を指導することはできない。

そこで，本研究では，プログラムの記述が変わると，プログラムの挙動がどのように変化するかということをプログラミングの初学者に理解させるために，プログラムの挙動を可視化して支援する，プログラミング学習支援環境AZURを開発して，プログラミングの演習授業に導入した。なお，AZURが対象とするプログラミング言語はプログラミングの教育で用いられることが多いC言語とした。

本稿では，AZURで採用している，プログラムの挙動を可視化する方法を具体的に提案するとともに，実験を通じてAZURの可視化機能がプログラミングの学習に有効であることを示す。

本稿の構成を以下に示す。2章では，最初に，プログラミングの初学者がプログラムの挙動を正しく想い描くことができない理由を明らかにする。しかる後に，プログラミングの初学者がプログラムの挙動を正しく想い描くことができるようにするためには，制御構文と再帰関数の挙動を正しく思い描けるようにすることが必要だということを述べる。3章では，プログラムの挙動を示すツールの研究を列挙して，本研究が目指すところとの違いを明らかにする。4章では，入れ子になった制御構文と再帰関数の挙動を，

†1 国立情報学研究所 先端ソフトウェア工学・国際研究センター
National Institute of Informatics
†2 (株)ビーブレイクシステムズ
bBreak Systems Co. Ltd.
†3 日本工業大学
Nippon Institute of Technology
†4 芝浦工業大学
Shibaura Institute of Technology

学習者に正しく思い描かせることを目的とした，プログラミング学習支援環境AZURが持つべき機能を明らかにする。5章では，AZURが持つ機能と，AZURが採用する，プログラムの挙動を可視化する方法を具体的に述べる。6章では，実験を通じてAZURがプログラミングの学習に有効であることを示す。7章では，本稿のまとめを述べる。

2. 初学者がプログラムの挙動を正しく思い描けない理由

初学者がプログラムの挙動を正しく思い描くのが困難な構文として，本稿ではプログラムの処理の流れを制御する構文(例えば，if 文，for 文などの制御構文)と，再帰関数を採り上げる。

初学者は，プログラムに含まれる制御構文が一つだけなら，プログラムの挙動を正しく思い描くことができるが，制御構文が入れ子になると，プログラムの挙動を正しく思い描くことができなくなるのではないかと推測される状況が，演習授業において観測された。その原因として下記の二つが考えられる。

一つ目の原因は，プログラムの各構文が，プログラムとしてどのように挙動するかということが一般的な教材では詳しく説明されていないことにある。特に，入れ子になった制御構文の挙動については，ほとんど説明がない。そのため，プログラムの処理結果のみを頼りにしてプログラムの挙動を思い描くので，正しく思い描くことができない。

二つ目の原因は，制御構文の制御範囲を初学者が正しく識別できないことにある。制御範囲とは，例えば，for文で繰り返して実行されるのは，どの行からどの行までの範囲か，などを指す。

多くの初学者はプログラミングスタイルには関心が低いので，制御構文が入れ子になった場合には，インデントが正しくないことが多い。また，たといインデントが正しかったとしても，プログラムの挙動を思い描く際には制御範囲以外にも注目するべき部分が多いので，制御範囲に対する注意が疎かになり，制御範囲を誤って識別してしまうことが少なくない。制御範囲を誤って識別していれば，たといプログラムの実行軌跡を示したとしても，プログラムの挙動を正しく思い描くことができない。

これらの問題点を解決するには，複雑な構文でも正しく識別できるように，その構文の制御範囲を強調して示すとともに，その構文に対応するプログラムの挙動を示す必要がある。

しかし，プログラムの挙動を示すだけでは，構文の記述内容とプログラムの挙動とを対応付けることができず，何故そのような挙動になったか理解できない。このため，いつまでたってもプログラムの挙動を正しく思い描けるようにならない可能性がある。そのため，構文の記述内容と同時に，対応するプログラムの挙動との対応関係を理解できるような形で示す必要がある。

再帰関数については，再帰呼び出しが行われると，どこに処理が移るのか，そして関数の処理を終えた後，どこに戻ればいいのかということが全く理解できない。その原因は，一般的な教材では再帰関数の挙動について，ほとんど説明がないからである。また，再帰関数に対しては，関数内で宣言された変数などがスタックに積まれ，return文が実行されるとスタックを解放し，実行元に戻り値を返すという，関数呼び出しの際の一連の挙動を示すことが，プログラムの挙動を正しく思い描くために必要だと考えられる。

3. 関連研究

プログラムの挙動をプログラムの実行軌跡で示すツールとしては，GDB[2]やVisual Studio[7]のようなシンボリックデバッガがある。デバッガは，行単位でプログラムの実行軌跡を示すことができる。しかし，構文の記述内容と実行軌跡とを対応づけて示すことはしていない。また，関数呼び出しの際の一連の挙動を示すこともしていない。そのため，デバッガは学習者が制御構文に対応するプログラムの挙動を正しく思い描くことには寄与していない。

Jeliot3[8]はJava用のプログラム可視化ツールである。Jeliot3はプログラムを式単位で実行することができ，その式の評価過程や，オブジェクトの参照関係，関数呼び出しの過程を実行軌跡の形で示す。このため，式単位での実行はfor文の挙動を理解させるのに有用である。しかし，Jeliot3ではすべての可視化が一塊になっているため，必要な可視化と不必要な可視化とが混在してしまう。これは利用者にとって好ましいものではない。

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はAvis[5]がある。Avisは学習者が記述したプログラムから，取りうる実行経路の一覧をフローチャートの形で出力する。しかし，プログラムとフローチャートとを対応させ，制御構文の制御範囲を強調して示すようなことはしていない。そのため，制御構文の制御範囲の誤りを正すことはできない。

新開ら[9]はプログラムのアルゴリズムを逐次，選択，

反復の3つからなる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をステップ実行するツールを開発している。制御構文が入れ子になった場合には，それをツール上では包含関係を繰り返す形で表示するため，各制御構文の制御範囲をそれぞれ明確に示すことができる。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ので，実際のプログラムとの対応も確認することができる。しかし，擬似言語上では選択処理はif文で，反復処理はwhile文でしか表現できない。このため，switch，do-while，for文などの挙動は正しく思い描くことができない。

関数の挙動を理解させることを目的としたツールとして，ETV[10]がある。ETVはプログラムを行単位で実行し，関数呼び出しが行われると，現在のソースコード表示画面の横に，新たなソースコード表示画面を作成する。そのため，関数の呼び出しが終了した後，どの行に戻ってくるかが理解できる。しかし，ETVでは関数呼び出しの際のスタックフレームに対する一連の挙動を示すことまではしていない。

松田ら[6]は再帰関数の挙動を理解させること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ーにプログラム部品の組み合わせをさせることで再帰関数を表現させている。ツールは，作成されたプログラムの中で変数の値がどのように変化して行くのかをアニメーションで表現する。変数の値が変化して行く過程を履歴の形で一覧できるので，再帰呼び出しの度に変数の値が変化して行く過程を理解させることができる。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方式上の制約があるため，作成可能なプログラムは限られている。

これまで述べてきたように，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，制御構文が入れ子になったときの，様々な制御構文に対応する，プログラムの挙動を学習者に思い描かせるツールは未だ存在しない。また，関数呼び出しの際の，スタックフレームが積まれ，開放されるまでの一連の挙動を示すツールも未だ存在しない。

4. 提案する可視化の方法

長谷川ら[3]は，初学者に対して，制御構造を含んだプログラムの動作の理解度と，制御構造に対する動作イメージとの関係を調査した。その結果，次の2点を明らかにした。

(1) プログラムの動作イメージを持たない初学者よりも，動作イメージを持っている初学者のほうがプログラムの動作を正しく理解している。

(2) より抽象的な動作イメージを持っている初学者のほうが，そうでない初学者よりもプログラムの動作を正しく理解している。

例えば，初学者はfor文に関して，ループ型，階段型，例示型などの動作イメージを持っており，ループ型の動作イメージを持つ初学者は，他の動作イメージを持つ初学者よりもプログラムの理解度が高い。

以上の調査結果を考察すると，階段型や例示型の動作イメージを持つ初学者は，制御構造がネストした場合には，対応する動作イメージを作成できないので，実行過程を正しく理解できないのだと考えられる。このことから，階層的な表現が可能な動作イメージを持てるような方法で，制御構文の制御範囲を示すことが，ネストした制御構文の理解には効果的であると考えられる。

再帰関数については，初学者の多くは関数のスタックフレームを理解しておらず，関数の動作イメージを持てていないと考えられる。また，関数呼び出しの際には，ただ単に実行行が変化するだけとしか考えていない初学者が多い。関数呼び出しの際に行われる，関数内で宣言された変数と戻り番地がスタックに積まれ，return文が実行されるとスタックを解放し，実行元に戻り値を返すという，一連の動作を示すことで，再帰関数の動作イメージが作成され，再帰関数の実行過程を理解できると考えた。

そこで，制御構文の制御範囲と再帰関数を呼び出したときの振る舞いを可視化することにした。その一方で，プログラムの静的な構造をハッキリと示し，その上でプログラムを逐次実行させることにより，プログラムの実際の実行過程を示すことが重要であると考えた。しかし，プログラムの実行過程を示すだけでは，それが制御構文のどのような働きによって，そのような実行過程になったかを理解することは難しい。そこで，制御構文の働きを明らかにするために，プログラムの実行が今どの制御構文の中にあり，その構文を今から実行しようとしているのか，終えようとしているのかを可視化することにした。

入れ子になった制御構文と再帰関数の挙動を，学習者に正しく思い描かせることを目的とした，プログラミング学習支援環境AZURを提案する。2章で述べた，初学者がプログラムの挙動を正しく思い描くことができない理由を踏まえ，AZURが持つべき機能を以下に列挙する。

(F1) 制御構文のブロック構造の可視化

初学者の注意が疎かになっている，制御構文の制御範囲を強調して示す。制御構文が入れ子になったときにも，それぞれの制御範囲を正しく認識できるように，包含関係を繰り返す形で可視化を行う。

(F2) 制御構文や再帰関数の挙動と実行行との対応付け

(ア)プログラムの挙動と実行行との対応付

入れ子になった制御構文の挙動を明らかにするために逐次実行させ，今まさに実行しようとしている行がどの制御構文の中のどこにあり，その構文を今から実行しようとしているのか，終えようとしているのかなども可視化する。

(イ)スタックフレームの可視化

再帰関数の呼び出しと戻りの挙動を理解させるために，スタックフレームの可視化を行う。

5. プログラミング学習支援環境 AZUR

5.1 AZUR の機能

5.1.1 制御構文のブロック構造の可視化

1 つの制御構文が，どの行から始まり，どの行までで，制御構文として意味のある塊(この制御構文のブロック)となるのかを， 図 1 のように開始行の字下げ位置と終了行のそれとを同一にする形で示すことにより，1 つの制御構文の制御範囲を示す。

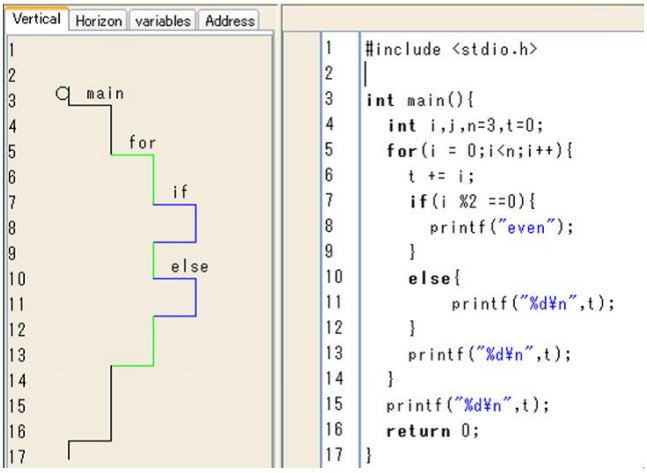

図 1 制御構文のブロック構造の可視化

このとき，入れ子になった制御構文の場合，例えば，制御構文 A の制御範囲の中に制御構文 B を含み，制御構文 B の制御範囲の中に制御構文 C を含むような場合には，次のように図示する。

制御構文 A の開始行の字下げ位置よりも，制御構文 B の開始行の字下げ位置を下げ，制御構文 B の終了行の字下げ位置を制御構文 B の開始行の字下げ位置と同一にする。そして，制御構文 B の開始行の字下げ位置よりも，制御構文 C の開始行の字下げ位置を下げ，制御構文 C の終了行の字下げ位置を制御構文 C の開始行の字下げ位置と同一にする。

しかも，これらの字下げの状況を，ソースコードの行に合わせて図示する。このようにすることによって，ソースコードの制御構文が入れ子になっている場合でも，図示されている字下げ位置を見るだけで，入れ子の状況とそれぞれの制御構文の制御範囲が判るようにする。

しかし，初学者は，図示されている字下げ位置を見るだけでは，制御構文の制御範囲を識別できない可能性もあるので，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図示し，それぞれが意味の違うブロックであることを示した。なお，これらの図示表示は，ユーザーが行うソースコードの書き換えに追随してリアルタイムで更新される。

switch 文の case，default 節は文法上ではブロックとは見なされないが，switch 文の制御範囲を明確にするために，通常のブロックと同じように図示している。また，break 文がないために fall through が発生するブロックに対しては，ブロックの終わりを示す線を破線（図 2 中の case:1 ブロックの末尾部分）で表示することにより，fall through が起こることを示している。

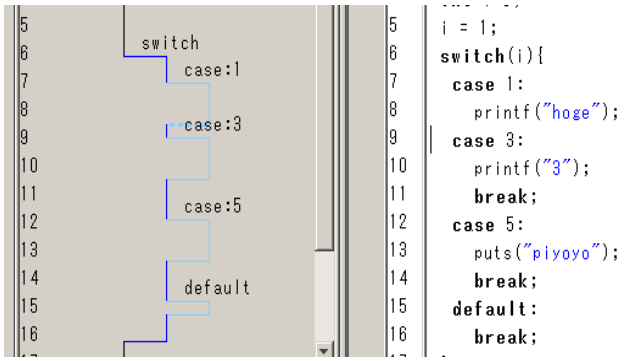

図 2 switch 文の可視化

5.1.2. 実行過程と構文の働きの対応表示

プログラムの挙動を行単位で示すために，行単位で行われるプログラム実行(ステップ実行)の軌跡を，今まさに実行中の行(プログラムカウンタの所在地に相当する)をボールで示し，このボールが移動して行く様を，プログラム実行の軌跡として図示することによって示す。

図 3 は，「今まさに for 文のブロックの中の処理に突入するかどうかを判定する処理の行にプログラムカウンタがある」ということを，ボールを用い

て表現している。

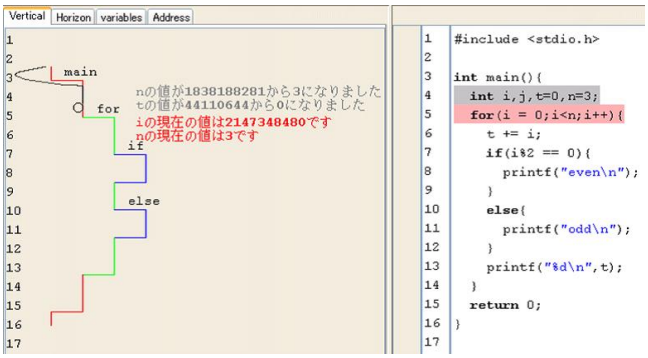

図 3　ボールを用いた，今まさに実行中の状態にある行の表示

また，このボールの移動の軌跡を表示し，消さずに残している。こうすることにより，反復構文が同じ箇所を何度も通る状況と，関数呼び出しからの復帰場所が視覚的に判りやすくしている。

ボールの横には，次に実行する行で参照する変数の値と，直前のステップ実行で変化した変数の値を表示する。これにより，利用者が注目する部分を少なくすることにより，注目する部分をより明確にすることができる。利用者はプログラムの制御範囲に注目しつつ，条件式で利用されている変数の値にも注目することができる。

5.1.3 スタックフレームの可視化

再帰関数のように，ある関数が複数回呼び出された時は，図 4 のようにその関数のブロック構造を複製して順次右側に表示することで，スタックフレームが積まれて行く様子を表現する。関数の処理が終わったら，複製されたブロック構造の表示を消去することで，スタックが開放されたことを表現する。

関数に戻り値がある場合には，ボールに色をつけることで戻り値があることを示し，その値をボールの横に表示することで，戻り値を表現する。

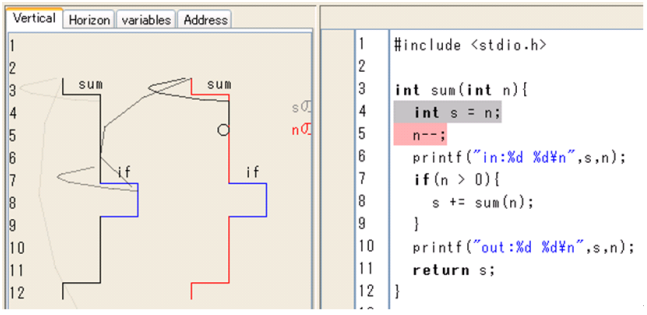

図 4 再帰関数の可視化

5.2 AZUR の構成

AZUR は Java の IDE として実装され，バックエンドで GCC[1]，GDB を用いることでプログラムのコンパイル及び実行を制御する。AZUR に対してユーザーが行った操作については随時ログサーバに送信され，蓄積が行われる。

現時点での AZUR は配列，構造体，ポインタについては，GDB からの出力をそのまま表示するだけに留め，可視化していない。しかし，ブロック構造の可視化を行っている部分は，タブを用いて表示しているため，新しい可視化を容易に追加できる。タブを切り替えることで，利用者は自身が必要とする要素の可視化のみを見ることができる。

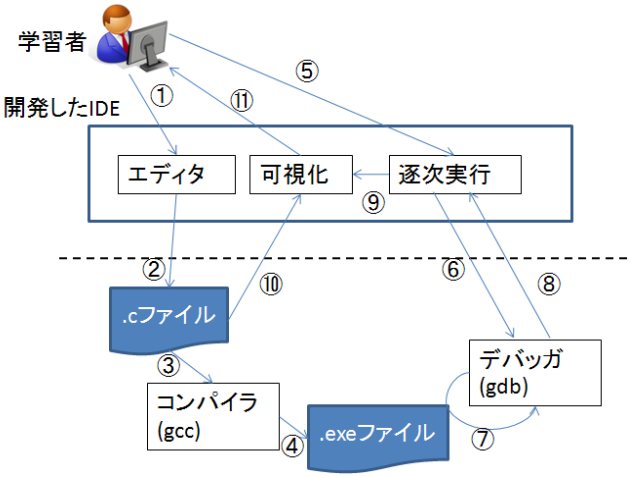

図 5 AZUR の構成

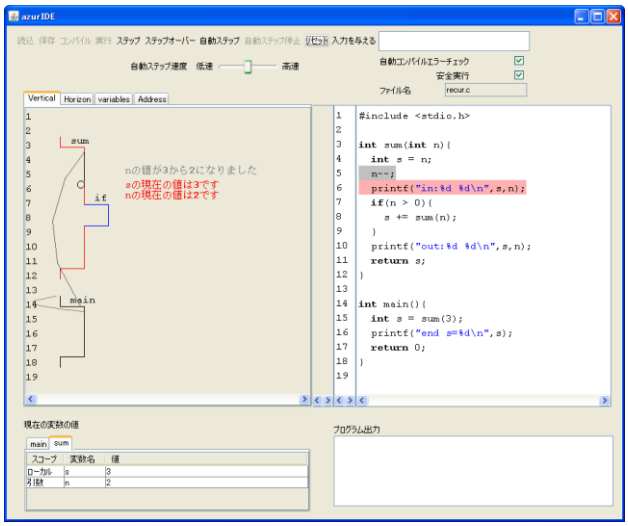

図 6 AZUR の画面全体

6. AZUR の可視化機能がプログラミングの学習に有効であることの確認

6.1 AZUR の使用効果を確認するための実験

芝浦工業大学デザイン工学科 2 年生を対象とするプログラミングの演習授業(前期の授業)に初めて AZUR を導入し，受講者全員に使ってもらった。（AZUR の使用方法は演習授業の中で説明した。）

AZUR はエディタの機能(ただし，インデントを自動修正する機能はない)も持ち合わせているので，学生は AZUR を使用して，プログラミングのためのすべての作業を行った。

AZUR を使用したことが学習に有効であったかどうかを確認するための実験を行った。なお，受講者は既に 1 年生の後期に C 言語の講義(1 コマ)と C 言語のプログラミングの演習(1 コマ)という連続授業(計 180 分)を週 2 回，15 週間受講している。

・ 被験者

C 言語の講義(1 コマ)と C 言語のプログラミングの演習(2 コマ) という連続授業の履修者 41 名

・ 実験手法

(1) 演習授業の初回に被験者全員に対して実力テストを行った

(2) AZUR を使用して，C 言語の講義(90 分)と C 言語のプログラミングの演習(180 分)という連続授業(計 270 分)を 15 週間行った(同時期に C 言語やプログラミングに関する授業はない)

(3) 演習授業の最終回に，初回と同じ問題を出題し，正答数の変化を調べた

・ 実力テストの問題

入れ子になった if 文，入れ子になった for 文，再帰関数をそれぞれ含む 3 つのソースコードを用意した。被験者に対し，これらのプログラムを実行したときの実行行の変化とそのときの変数の値の変化を図 7 のように記述させた

問題

```
1  int main(){
2    int i,j,n=3,t=0;
3    for(i=1;i<n;i++){
4      t+=I;
5      for(j=1;j<n;j++){
6          t+=j;
7      }
8      printf("%d¥n",t);
9    }
10   printf("%d¥n",t);
11 }
```

解答例

実行行	変数の値			
	i	j	n	t
2			3	0
3	1			
4				1
5		1		
6				2
7				
5		2		
6				4
...				

図 7 実力テストの問題と解答例

この実験から得られた, AZUR を使う前と使った後における正答者数と誤答者数の変化を調べて 2×2 分割表(偶然表)にまとめたものが表 1～表 4 である。実力テストに用いたソースコードは 3 種類であるが，分析は if 文の場合，for 文の場合，再帰関数の場合，制御範囲の 4 つの観点から行った。

これらの表の読み方を表 1 の場合(if 文の場合)を例に説明する。41 名の被験者が if 文の問題に挑戦したら，AZUR を使用する前には 25 名が正解し，16 名が不正解した。そして，AZUR を使用して学習したら，41 名中の 35 名が正解し，6 名が不正解した。つまり，被験者 41 名のうち，25 名は AZUR を使用する前から正解であり，10 名は不正解から正解に転じ，6 名は不正解のまま(変化なし)であった，と読む。

表 1　AZUR の使用による 正答者数と誤答者数の変化 (if 文)

	AZUR 使用前	AZUR 使用後	横方向の合計
誤答者の数	(a) 16	(b) 6	(a+b) 22
正答者の数	(c) 25	(d) 35	(c+d) 60
	(e) 0	(f) 10	(e+f) 10
縦方向の合計	(a+c) 41	(b+d) 41	(a+c+b+d) 82
	(a+e) 16	(b+f) 16	(a+e+b+f) 32

表 2 AZUR の使用による正答者数と誤答者数の変化 (for 文)

	AZUR 使用前	AZUR 使用後	横方向の合計
誤答者の数	(a) 36	(b) 16	(a+b) 52
正答者の数	(c) 5	(d) 25	(c+d) 30
	(e) 0	(f) 20	(e+f) 20
縦方向の合計	(a+c) 41	(b+d) 41	(a+c+b+d) 82
	(a+e) 36	(b+f) 36	(a+e+b+f) 72

ところで，表 1～表 4 では，「正答者の数」の欄がいずれも上下 2 段になっていることを奇異に思われる方もおられるかと思う。これを表 1 の場合で説明する。被験者 41 名のうちの 25 名は AZUR を使用する前から正解だったので，この 25 名は AZUR 使用効果の有無とは無関係だと考えたのが，「正答者の数」の欄の下段である。すると被験者は 16 名となり，このうちの 10 名が不正解から正解に転じ，6 名は不正解のまま(変化なし)であった，と解釈できる。これに

対して，AZUR を使用する前から正解だった 25 名を被験者から外すことなく，そのまま扱ったものが「正答者の数」の欄の上段である。「縦方向の合計」の欄は，この考えを反映したもので，上段が AZUR を使用する前から正解だった 25 名を被験者に含めた場合の合計で，下段が含めない場合の合計である。

表 3 AZUR の使用による正答者数と誤答者数の変化（再帰関数）

	AZUR 使用前	AZUR 使用後	横方向の合計
誤答者の数	(a) 39	(b) 29	(a+b) 68
正答者の数	(c) 2	(d) 12	(c+d) 14
	(e) 0	(f) 10	(e+f) 10
縦方向の合計	(a+c) 41	(b+d) 41	(a+c+b+d) 82
	(a+e) 39	(b+f) 39	(a+e+b+f) 78

表 4 AZUR の使用による正答者数と誤答者数の変化（制御範囲）

	AZUR 使用前	AZUR 使用後	横方向の合計
誤答者の数	(a) 21	(b) 6	(a+b) 27
正答者の数	(c) 20	(d) 35	(c+d) 55
	(e) 0	(f) 15	(e+f) 15
縦方向の合計	(a+c) 41	(b+d) 41	(a+c+b+d) 82
	(a+e) 21	(b+f) 21	(a+e+b+f) 42

6. 2 検定の方法

AZURを学習に使用したことが，正答者と誤答者の数に関して，統計的に有意な変化をもたらしたかどうかの検定を，これらの表を基に行う。

表1～表4は，AZUR使用前とAZUR使用後，誤答と正答という，いずれも二律排反の事象を表している。これから行うのは，AZUR使用前とAZUR使用後という事象と，誤答と正答という事象とは互いに独立であるか（つまり，誤答と正答という事象は，AZUR使用前とAZUR使用後という事象とは無関係であるか）どうかを検定するものである。

このとき，下記のような帰無仮説(H0)と対立仮説(H1)を立てて χ^2 検定を行う。

H0: AZUR使用前とAZUR使用後という事象と，誤答と正答という事象とは互いに独立である

H1: 独立ではない
(AZURを使用した効果が認められる)

6. 3 χ^2 値の求め方

6. 3.1 41 名全員を被験者と考える場合

(1) 分割表の中に度数の小さい値(5 以下)が含まれていない場合(補正が必要でない場合)，下記の式を用いてカイ二乗値を計算する。

$$\chi^2 = \frac{(a+b+c+d)(ad-bc)^2}{(a+b)(a+c)(b+d)(c+d)} \quad (1A)$$

(2) 分割表の中に度数の小さい値(5 以下)のものが含まれている場合(補正が必要な場合)，上記の計算式では近似精度が悪くなるので，Yates の補正 [4] と呼ばれる次式を用いてカイ二乗値を計算する。

$$\chi^2 = \frac{(a+b+c+d)\{|ad-bc|-1/2(a+c+b+d)\}^2}{(a+b)(a+c)(b+d)(c+d)} \quad (1B)$$

6.3.2 AZUR の使用による正解者の増加数だけを被験者と考える場合

AZURを使用する前から正解だった者で，後に不正解となった者はいないので，「AZURの使用による正解者の増加数だけを被験者と考える」ことと，「AZURを使用する前から正解だった者は被験者に含めない」ことは同義である。

(1) 分割表の中に度数の小さい値(5以下)が含まれていない場合(補正が必要でない場合)，下記の式を用いてカイ二乗値を計算する。

$$\chi^2 = \frac{(a+e+b+f)(af-be)^2}{(a+b)(a+e)(b+f)(e+f)} \quad (2A)$$

(2) 分割表の中に度数の小さい値(5以)のものが含まれている場合(補正が必要な場合)，Yates の補正[4] と呼ばれる次式を用いてカイ二乗値を計算する。

$$\chi^2 = \frac{(a+e+b+f)\{|af-be|-1/2(a+e+b+f)\}^2}{(a+b)(a+e)(b+f)(e+f)} \quad (2B)$$

6. 4 検定の結果

6.3節で説明した方法で求めた χ^2 の値は表5のとおりである。このときの自由度 f は

$$f = (h-1)(k-1) = (2-1)(2-1) = 1$$

となるので，χ^2 分布表の有意水準(または危険率) α ＝ 0.05の数値を引くと，表5のいずれの場合の χ^2 値に対しても

$$\chi_{0.05}^{2}(1) = 3.84 < \chi^2$$

となるので，仮説(H0)は棄却され，対立仮説(H1)が採択される。よって，AZUR 可視化機能を使用して学習したことは，正答者数と誤答者数の変化(正答者数の向上)とは無関係(独立)ではない。すなわち，AZUR の可視化機能がプログラミングの学習に有効であったと結論できる。

表 5 計算によって求められた χ^2 の値

被験者の範囲	41 名全員を被験者とする		AZUR を使用する前からの正解者を除く	
使用した計算式	1A	1B	2A	2B
if 文	6.212	~~5.032~~	~~14.545~~	11.782
for 文	~~21.026~~	18.976	~~27.692~~	24.992
再帰関数	~~8.613~~	6.977	~~11.471~~	9.291
制御範囲	12.424	~~10.823~~	~~23.333~~	20.326

7. むすび

本稿では，最初に，プログラミングの初学者がプログラムの挙動を正しく想い描けない理由を明らかにした。しかる後に，プログラミングの初学者がプログラムの挙動を正しく想い描くことができるようにするためには，制御構文と再帰関数の挙動を正しく思い描けるようにすることが必要だということを述べた。そして，プログラムの挙動を示すツールの研究を列挙して，本研究が目指すところとの違いを明らかにした。しかる後に，入れ子になった制御構文と再帰関数の挙動を，学習者に正しく思い描かせることを目的とした，プログラミング学習支援環境AZURを提案した。その中で，AZURが持つ機能と，AZURで採用する，プログラムの挙動を可視化する方法を具体的に明らかにした。最後に，実験で得られたデータに対して χ^2 検定を適用することにより，AZURの可視化機能がプログラミングの学習に有効であったことを示した。

なお, AZURはプログラミング学習支援機能だけでなく，プログラム開発支援機能をも併せ持った統合開発環境である。後者についての詳細は，別稿で明らかにしたい。

謝辞

プログラミングの演習授業へのAZURの導入を快く同意して下さるとともに, AZURを積極的に使用して戴き，いろいろご指導を賜った，芝浦工業大学デザイン工学部デザイン工学科の山崎憲一教授に深く感謝致します。

本研究は，文部科学省の科学研究費補助金基盤研究（C）25330416「摂動に基づく，プログラミング言語の文法知識習得支援技術の研究開発」の援助を受けている。

参考文献

1) Free Software Foundation, Inc. (2012). GCC: the GNU Compiler Collection 2012年1月24日 〈http://gcc.gnu.org/〉(2012年02月27日)

2) Free Software Foundation, Inc. (2012). GDB: The GNU Project Debugger 2012年2月27日 〈http://www.gnu.org/software/gdb/〉(2012年2月27日)

3) 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, 名古屋文理短期大学紀要, Vol.22, pp.9-14 (1997).

4) 岩原信九郎: 教育と心理のための推計学(新訂版) 日本文化科学社, (1965).

5) 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).

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 電子情報通信学会和文論文誌D1, Vol.80, No.1, pp. 326-335 (1997).

7) Microsoft Corporation(2012). Microsoft Visual Studio 2012年 〈http://www.microsoft.com/visualstudio/〉(2012年02月27日現在)

8) Moreno, A., Myller, N. , Sutinen,E. and Ben-Ari M. Visualizing programs with Jeliot3 Proc. Advanced Visual Interfaces, pp.284--290(2004).

9) 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).

10) Terada M.: ETV: a program trace player for students, Proc. of ITiCSE, Vol.37, No.3, pp. 118-122(2005).