Vol.2016-CE-136 No.4
2016/10/15

実行時の変数の値の変化のコード編集による変化の可視化

小山 秀明[†1,a)]　山田 俊行[†1,b)]

概要：プログラムを理解するためにはプログラム実行時の変数の値の変化を理解する必要がある．しかし，プログラミング初学者にとって，ソースコードと実行結果から変化を理解することは困難である．また，ステップごとの変化を理解するだけではなく，プログラム全体を通した変化を理解することも重要である．１つの変数に着目したトレースとデータ依存を用いた可視化，ブロック毎の変数の値の変化の可視化，コード編集の前後の実行結果の比較，この３つの理解支援の手法を提案し， 理解支援ツールとして実装する．

キーワード：可視化，理解支援，初学者支援，データ依存，動作トレース

Visualizing The Change of Variable Value During Execution and The Change by Edit Code

Koyama Hideaki [†1,a)]　Yamada Toshiyuki[†1,b)]

Abstract: Understanding how the value of each variable changes during the program execution is necessary for program understanding. However, novice programmers have difficulty to understand the change from a source code and a result of program execution. It is important for program understanding to know not only change in each step but also change in multiple steps. We propose three methods for program understanding. The first method is using a trace and data dependence. The second method is visualizing the change of a value of a variables in a code. The thrid method is comparing the original program and the edited program. Moreover, we implement these methods as a tool.

Keywords: visualization, comprehension support, novice support, data dependence, action trace.

1. 背景

プログラミングにおいて，プログラムの動作を理解することは重要である．しかし，ソースコードとその実行結果からプログラムの動作を思い描くことはプログラミング初学者にとって困難である．その要因の１つとして，プログラム実行時の変数の値の変化が分からないことが挙げられる．

プログラム実行時の変数の値を知る方法の１つとして，GDB[1] のようなデバッガを用いる方法がある．しかし，デバッガの操作を覚える必要があり，ブレイクポイントの設定が必要なため，初学者にとってデバッガを使うことは困難である．そのため，初学者のためにプログラム実行中の変数の値を可視化するツールは有用であり，また，そのようなツールは既に多く存在している．

ステップ毎の変数の値を可視化することで，各ステップでの変数の値の変化が分かりやすくなる．しかし，プログラムの動作を理解するためには，ステップ毎の変化だけではなく，プログラム全体を通した変数の値の変化を理解することも重要である．多くの可視化ツールはステップ毎の可視化だけを行なっているため，プログラムを通した変化の理解に繋がらない．

本研究では，１つの変数の振舞いを抽出したトレースで

†1 現在，三重大学
Presently with Mie Uniersity
a) koyama@cs.info.mie-u.ac.jp
b) toshi@cs.info.mie-u.ac.jp

ある「動作トレース」にデータ依存を合わせた可視化，コード編集によるプログラムの動作の変化の可視化，コード編集による実行の変化の比較の３つのプログラム理解支援の手法を提案する．本研究での動作トレースは，１つの変数が関わったコードとその時の値から構成される．このトレースによりその変数がどのように使用されるかを示し，データ依存により変数の値を求める経路と変数間の関わりを示す．ブロック単位で変化を可視化することにより，１ステップずつの可視化では気付きにくい，まとまったコードによる変化に気付きやすくなる．ソースコードを１行編集した時のプログラムの動作の変化を比較させることで，そのコードの持つ意味の理解を支援する．これら３つの手法を理解支援ツールとして実装する．

2. 関連研究

Java プログラムの可視化ツールである Jeliot 3 [2] はプログラムを１ステップずつ実行する．１つ１つの処理をアニメーションで表現する．処理の実行，変数の変化が初学者でも分かりやすいが，プログラミングに慣れてくると丁寧過ぎるため冗長になってくる．１つの命令に対する動作は理解しやすいが，プログラム全体を通しての動作の理解にはつながり難い．

Avis[3] はプログラム自動可視化ツールである．ソースコードを読み込み、フローチャート，実行時の振舞いを示す逐次型実行経路図，モジュール間のつながりを示すモジュール遷移型実行経路図を作成し表示する．実行経路を示すことにより，プログラムの動作の理解支援を行うが，変数の値に対するサポートがないため，実行時の変数の値の変化は分からない．

ETV[4] はプログラムを一度実行し，その時の情報をトレースファイルとして記録したのち，そのトレースファイルを基にプログラムの実行を可視化するツールである．トレースを用いて実行を再現するため，実行のステップを進めるだけでなく戻すこともできる．現在停止している行をソースコード上で強調し，関数が呼び出された場合，呼び出された関数のコードを現在の関数のコードに重ねるように表示する．このように，関数のコールスタックをソースコードの重なりで示すことで理解を支援する．関数の呼び出しが分かりやすくなるが，関数内の動作の可視化はステップ実行による可視化しか行われていない．

AZUR[5] は C プログラムの学習支援環境である．分岐や繰り返しなどの制御文の範囲が明確になるようにインデントを線状に図示し，ボールが先の上を動くアニメーションを用いて逐次実行を示す．各ステップでの変数の値を表示することはできるが，複数ステップ間での変数の値は表示されない．

以上で述べたように，ステップ毎の可視化や，プログラム全体を静的な図として可視化するツールは存在しているが，プログラムを通した変化の理解の支援が十分ではない．また，コードを編集した時のプログラムの動作の変化に関する支援は行われていない．

3. 提案手法

本研究では，プログラムを通した変化の理解のための３つの手法を提案する．

- トレースとデータ依存を用いた可視化
- ブロック毎の変化の可視化
- コード編集による変化の比較

ステップ毎の可視化に加え，トレースとデータ依存を用いた可視化と，ブロック毎の変化の可視化を行うことで，ステップ毎の変化とプログラムを通した変化両方の，実行時の変数の値の変化を可視化する．

また，コード１行の編集の前後の実行結果を表示し，それらを比較させることで，そのコードがどのような意味を持つかの理解に繋げる．

これらの手法を実装することで，ステップ毎の変化，プログラムを通した変化，コード編集時の変化を可視化する理解支援ツールを実現する．

4. 実行時の変数の値の変化

4.1 ステップ毎の変化の可視化

ステップ毎の変化を示すために，変数とその値を表示し，参照された変数と代入された変数を強調表示する．ツールは１ステップずつ実行を進めながら，その時の変数の値の表示し，参照・強調された変数を強調する．ステップを戻せるため，一部の処理を繰り返し確認することもできる．

ツールの実行画面を図 1 に示す．左側にソースコードが表示し，現在停止している行を緑色で強調する．右側にその行で有効な変数とその値を表示し，参照・代入された変数を強調する．参照された変数は黄色で強調し，代入された変数は赤色で強調する．変数を強調することにより，その行に関わる変数が一目で分かるようになる．図 1 では，緑色で強調されている “x[j] = x[j - 1]” が実行され，x[1] と j が参照され，x[2] の値が１から５に変わっていることを示している．

4.2 動作トレースとデータ依存を用いた可視化

各変数の使われ方の理解を支援するために「動作トレース」を提案し，そのトレースとデータ依存を表示することで，値を求める経路を示し，変数間の関わりの理解を支援する．

4.2.1 動作トレース

本研究では，１つの変数の動作を抽出したトレースである「動作トレース」を提案する．動作トレースは，１つの変数に着目し，プログラム実行時にその変数がどのように使用されたかを抽出したトレースである．このトレースは，

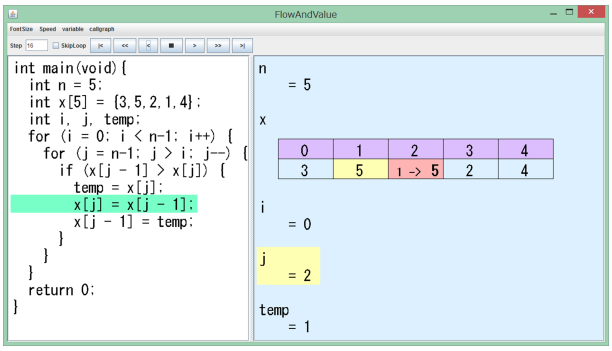

図 1 ステップ毎の可視化
Fig. 1 Visualizing the change of each step.

j		
参照	値	代入
for(j=n-1;j>i;j--){	4	for(j=n-1;j>i;j--){
if(x[j-1]>x[j]){	4	
for(j=n-1;j>i;j--){	3	for(j=n-1;j>i;j--){
if(x[j-1]>x[j]){	3	
temp=x[j];	3	
x[j]=x[j-1];	3	

図 2 動作トレースの表示
Fig. 2 Showing the action trace.

着目した変数が参照・代入された処理と，その時の変数の値から構成される．

表形式でトレースを表示することで，その変数が関わった処理の一覧を見せることができ，プログラム実行時にその変数がどのように使用され変化したかが分かりやすくなる．図 2 にツールが表示する表形式で表示されたトレースを示す．表は” 参照”，“値”，“代入” の 3 列から成る．それぞれ，その変数が参照された処理，その時の値，その変数に値が代入された処理を示す．これにより，その変数の値がどのように参照され，代入されたかが見やすくなる．着目した変数が配列の場合，参照されている要素を黒色，代入された要素を赤色，それ以外の要素を灰色で表示する．

配列を選択した時の動作トレースを図 3 に示す．このように，その処理に配列のどの要素が関わっているかが一目で分かる．また，表の中から変数を選択することで，その変数に関する表 (“値” と “代入” の 2 列から成る表) が追加で表示され，関係する変数の値がどう計算されたかを調べられる．１つの変数が関わった処理とその時の値の一覧を示すことで，プログラムを通して変数がどのように変化し，使用されたかが分かる．図 3 の値の列を見ると，参照・代入される要素が右から左へと移っていくのが分かる．これは，右端の要素から比較を行い，小さい値が左側へと入れ替えられていることを示している．

4.2.2 データ依存

動作トレースにより，１つの変数について処理を辿ることができる．さらに，データ依存を表示することで，複数表示された動作トレース間の関係を示し，変数間の関わりを示す．

x		
参照	値	代入
	35214	int x[5]={3,5,2,1,4};
if(x[j-1]>x[j]){	35214	
if(x[j-1]>x[j]){	35214	
temp=x[j];	35214	
x[j]=x[j-1];	35224	x[j]=x[j-1];
	35124	x[j-1]=temp;
if(x[j-1]>x[j]){	35124	
temp=x[j];	35124	
x[j]=x[j-1];	35524	x[j]=x[j-1];
	31524	x[j-1]=temp;
if(x[j-1]>x[j]){	31524	
temp=x[j];	31524	
x[j]=x[j-1];	33524	x[j]=x[j-1];
	13524	x[j-1]=temp;

図 3 配列の動作トレース
Fig. 3 The action trace of array

図 4 データ依存の表示
Fig. 4 Showing the data dependence.

図 4 は，一番下の行の配列 x の 0 番目の要素 (値”1”) のデータ依存を表示している．選択された値が依存している処理と値を背景を青色にして強調する．値から処理へ伸びる矢印は，その値が指し先の処理で使われていることを示す．強調部分と矢印を辿ることで，その値が求められた経路が分かる．また，その値を求めるために他の変数がどう関わったかが分かる．図 4 の強調された部分と矢印を辿ると，x の 4 番目の要素だった値”1” が，x の 0 番目の要素へと移ってきたことが分かる．

4.3 ブロック毎の変化の可視化

プログラムの構造毎のブロックを単位としてプログラム

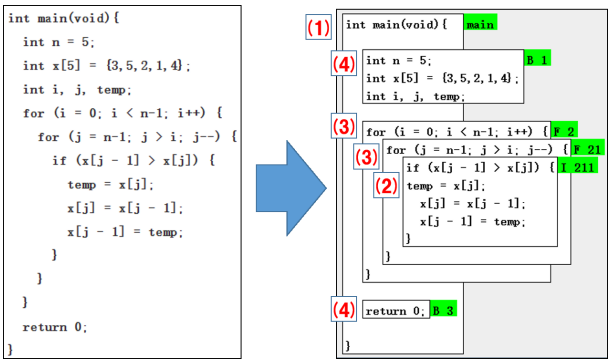

図 5 バブルソートのブロック
Fig. 5 Block of bubble sort

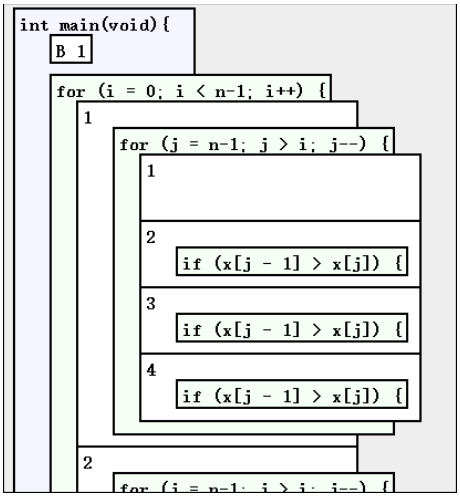

図 6 ブロックによる実行経路の表示
Fig. 6 Showing execution path by block

の実行を可視化する．ブロック単位で変数の値の変化を可視化することにより，１ステップずつ実行を追いかけるだけでは気付きにくい，ループ１回毎の変数の値の変化など，複数ステップによる変化に気付きやすくなる．

本手法では以下の４つをブロックとして扱う．

(1) 関数
(2) if などの分岐構造
(3) while などのループ構造
(4) 分岐やループの前後にあるコード

図 5 にバブルソートのソースコードを入力した時のブロックを示す．枠で囲まれた部分が１つのブロックである．ブロックの左側の数字は上記の (1)〜(4) に対応している．

このブロックを用いて実行経路を表示する．図 6 はバブルソートの外側の for ループの１回目の終了までを示している．for ブロックの中のブロックは繰り返しを意味しており，図 6 では内側の for ループが４回繰り返されたことを表している．また，分岐のブロックの場合は，実行されたブロックのみが表示され，条件式が偽となり実行されなかった場合，何も表示されない．

選択されたブロックの変数の値を表示する．そのブロックに入る時の値と，そのブロックから出る時の値の両方を表示し，ブロック内に代入がある場合，代入された変数の値を赤色で強調表示する．図 6 の内側の for ループの２回目を選択した時の表示を図 7 に示す．選択されたブロックが黄色で強調され，そのブロック内での変数の値が表示されている．左側に入る時の値，右側に出る時の値が表示され，代入された値が強調されている．これにより，ブロック単位での変数の値の変化が見られる．図 7 の配列 x の赤色で強調されている要素を見ると，入る時と出る時で値が入れ替わっていることが分かる．

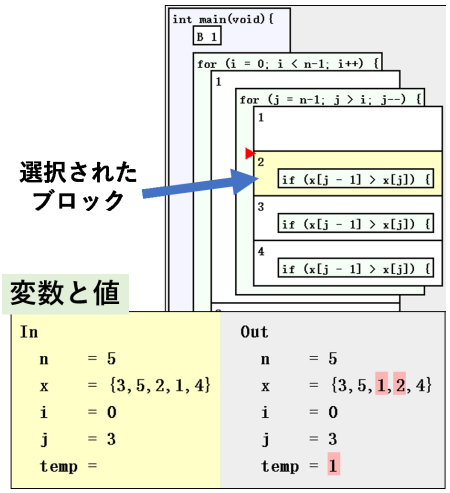

図 7 ブロック内における変数の値
Fig. 7 The values of variables in the block.

また，入る時または出る時の値が実行経路のどの位置での値であるかをブロックの左側の赤い三角形のマーカーで明示する．図 7 では入る時の値が黄色で強調されており，選択されたブロックの左上にマーカーがある．出る時の値に切り替えるとマーカーの位置がブロックの左下に変わり，出る時の値が強調される．

ツールの実行画面を図 8 に示す．左上にソースコードのブロック，左下に変数の値，右側にブロックによる実行経路が表示されている．ソースコードのブロック上のマーカーは，実行経路上で選択されたブロックのマーカーの位置に対応したソースコードを示している．

5. 編集による変化の可視化

１行のコード編集によるプログラムの実行時の動作の変化を可視化することで，その行がプログラムにどのような影響を与えているかを示す．例えば，ソートアルゴリズムの入れ替えの条件式の不等号を逆にすると，ソートの結果が逆順になる (昇順であれば降順になる)．これにより，入れ替えの条件式がソートの向きに関わっていることが分かる．このように，実際にコードを編集し，その結果を比較することで，コードの持つ意味の理解に繋げる．また，この値を変えるとどうなるだろう，この式を変えるとどう

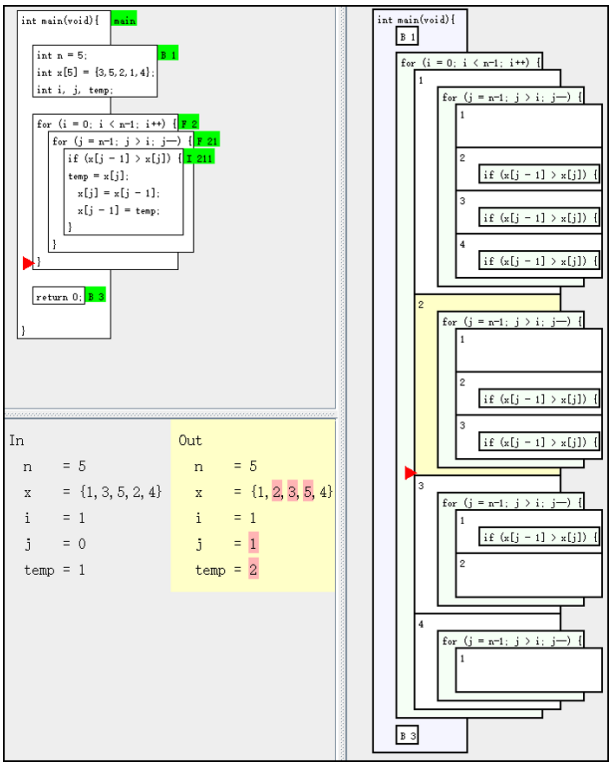

図 8 ブロック毎の可視化
Fig. 8 Visualizing the change of each block

なるだろう，といった学生の疑問に対して即座に答えを示せる．

図 9 に，バブルソートの入れ替えの条件式の不等号を逆にした時の，編集前と編集後の両方の実行結果を示す．左側が編集前の実行結果，右側が編集後の実行結果である．左下と右下のそれぞれの x の値を見ると，数字が逆順に並んでいることが分かる．また，実行経路を比較し，違いのあるブロックを選択することで，その時の変数の値を比較できる．

6. ツールの実装

本研究では，提案手法をツールとして実現した．ツールは C 言語のプログラムを入力として受け取る．様々なプラットフォームでの実行を可能にするため，ツールは Java 言語で実装されている．GDB??を用いてプログラムを実行することで，プログラム実行時の情報を取得する．取得する情報は，停止した行と，その行で更新された変数の値である．本ツールでは，ツール起動時にプログラムを最後まで実行し，予め最初から最後までの実行情報 (実行トレース) を取得する．この実行トレースを用いてプログラムの実行を再現し，可視化を行う．

6.1 ステップ毎の可視化

ステップ毎の変数の値の可視化では，実行トレースを用いて 1 ステップ毎の可視化を行う．ユーザの操作にあわせて実際にプログラムを 1 ステップずつ実行しているのではなく，既に取得している実行トレースを用いて実行を再生している．そのため，ステップを進めるだけではなく，ステップを戻す，任意のステップへのジャンプが可能である．

6.2 動作トレースとデータ依存

実行トレースから着目する変数が関わった行だけを抽出することで動作トレースを生成する．データ依存の取得は，プログラムの静的解析は行わず，実行トレースを辿ることで取得する．

6.3 編集による変化の可視化

編集されたソースコードをファイルに書き込み，再度 GDB でプログラムを実行する．元のプログラムと同様に，プログラムを最後まで実行し，実行トレースを取得する．そして，元の実行トレースと比較させる．

6.4 プログラムの制限

可視化する前ににプログラムを最後まで実行する必要がある．そのため，無限ループがあり停止しプログラムなど，一定のステップ数を超えるプログラムは途中で実行を打ち切り，その時点までの実行トレースを用いて可視化を行う．また，現時点では，ポインタや構造体などの複雑なデータ構造には対応しておらず，可視化できない．

7. 結論と課題

1 つの変数に注目したトレースである「動作トレース」を提案し，そのトレースとデータ依存を用いて，プログラム実行時の変数の値の変化の理解を支援する手法を提案した．動作トレースにより，その変数がどのように参照・代入されたかを示す．データ依存を表示し，変数の値が求められた経路と変数間の関係を示す．ステップ毎の変数の変化の可視化と，トレースとデータ依存を用いた可視化により，ステップ毎の変化とプログラムを通した変化の両方の理解支援を行う．また，ブロック単位で実行を制御することにより，ブロック毎の変数の値の変化の可視化することで理解を支援する手法を提案した．変数の値の変化をステップ単位ではなく，複数のステップからなるブロック単位で可視化した．

ソースコードを 1 行編集した時のプログラムの編集前と編集後の両方の実行結果を可視化することで，ユーザに実行結果を比較させることで各行の振舞いの理解を支援する手法を提案した．

これらの手法を実装することで，ステップ毎の変数の値の変化，プログラムを通した変数の値の変化，ソースコード編集によるプログラムの動作の変化を可視化するツールを実現した．

今後の課題として，ステップ毎の変化，プログラムを通

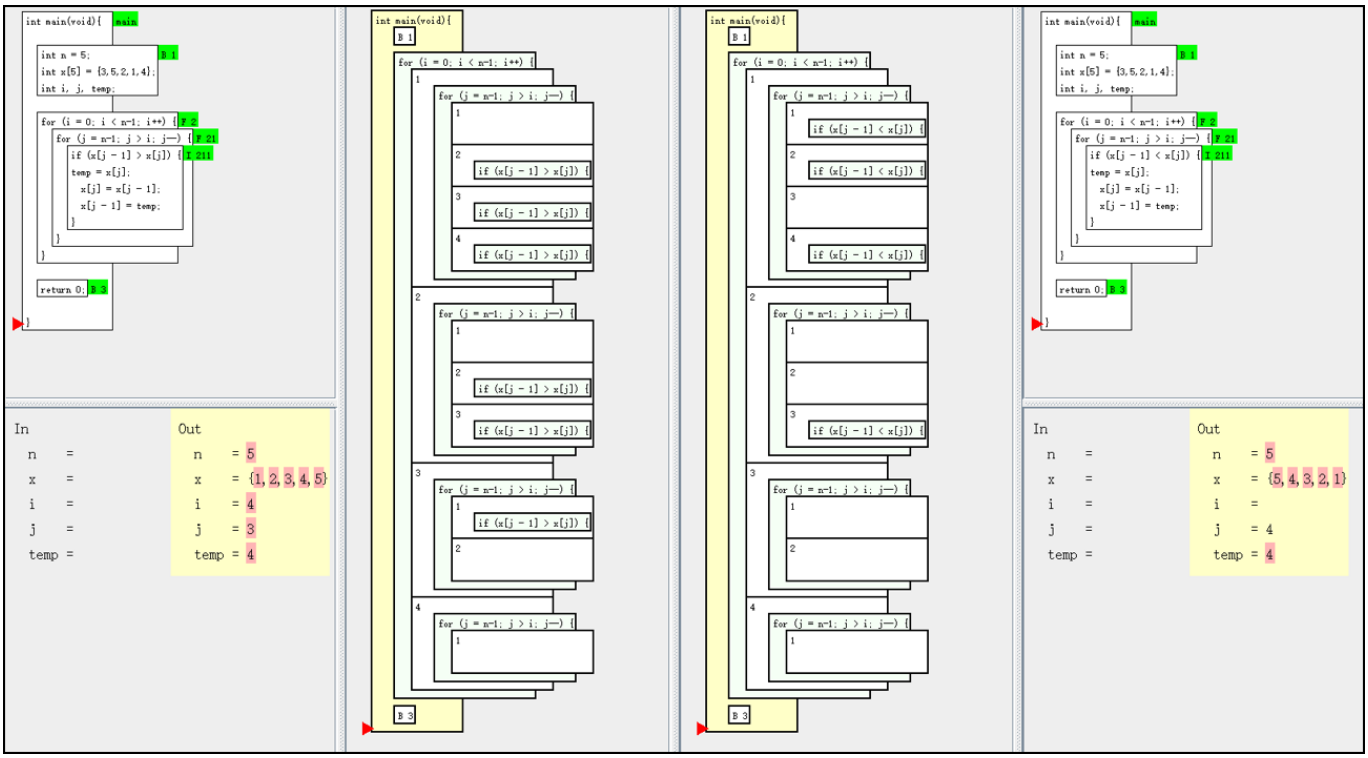

図 9 実行の比較
Fig. 9 Compare executions

した変化，編集による変化，それぞれの可視化をより深く関連付けていく必要がある．それぞれの可視化の関係の理解をサポートすることで，より良い理解支援が行えると考える．また，現時点ではポインタや構造体などの複雑なデータ構造は扱えないが，特にポインタは初学者にとって理解し難いデータ構造のため，ポインタを扱えることで支援の幅が広がると考える．

参考文献

[1] GDB: The GNU Project Debugger, http://www.gnu.org/software/gdb/.
[2] A. Moreno, N. Myller, and E. Sutinen, “Visualizing Programs with Jeliot 3”, Proceedings of AVI ’04, pp. 373–376, 2004.
[3] 喜多義弘，片山徹郎，冨田重幸 「Java プログラム読解支援のためのプログラム自動可視化ツール Avis の実装と評価 (ソフトウェアシステム)」，電子情報通信学会論文誌. D，情報・システム J95-D(4)，pp. 855–869，2012.
[4] M. Terada, “ETV: a Program Trace Player for Students”, Proc. 10th Conference on Innovation and Technology in Computer Science Education, pp. 118–122, 2005
[5] 古宮誠一，今泉俊幸，橋浦弘明，松浦佐江子，「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—」，情報処理学会研究報告，ソフトウェア工学研究報告，Vol. 2014