初学者向けのポインタを用いたプログラムの動作理解支援方法の提案

2012SE105 河本健吾　2012SE279 山田恭裕
指導教員：蜂巣吉成

1　はじめに

プログラミング学習において学習者は，ソースコードと実行結果からプログラムの動作把握をし，ソースコードの修正を繰り返すことによりプログラミング技術を習得している．プログラムの動作把握を行う際に，コンパイルエラーはないが意図した実行結果になっていない場合，ソースコードを見直し，修正する必要がある．しかし，初めてプログラミングにふれる学習者が，実行結果を確認するだけで動作理解することは難しい．

初学者はプログラミング学習の中でもポインタの学習でつまづきやすい．今まで，記述そのままの変数を参照していたものが，ポインタ変数によって指したメモリアドレスに格納されている値を参照するといった間接的で複雑なものになり，初学者の混乱を招きやすい．また，ポインタ変数の初期化部分の記述により，初学者はアドレス演算子 (&) と間接演算子 (*) の意味を混同してしまいがちである．そういった初学者は関数の引数などにポインタが関係してくると，参照関係を上手く把握することができず混乱しやすい．初学者からポインタの混乱を取り除きプログラムの動作をイメージさせることが重要である．特に間違えたソースコードを書いてしまった際，ソースコードを修正するためにプログラムの動作を正しくイメージすることが重要となる．

本研究では，変数の値の変化とポインタ，関数呼び出しを可視化し，初学者のプログラム動作理解を支援するためのツールを提案する．本研究は，先行研究 [1] で提案されている動作理解支援ツールを利用して，変数の値の変化を表 (以下，変数値履歴表と呼ぶ) で可視化し，ポインタについては，利用者が変数値履歴表から選択したポインタ変数と関連している変数の関係図 (以下，変数関連図と呼ぶ) を作成する．利用者が変数関連図を見ることによって，ポインタの状態を視覚的に確認できるようにする．また，本ツールで誤ったプログラムも可視化することによって理解支援を行えるか考察する．

2　関連研究

文献 [2] で提案されているツールは，ポインタを使用したプログラムの可視化を行い，ポインタの参照方法の理解支援を行う．GDB[6] でステップ実行を行いながら各変数の値やアドレスなどが書かれたメモリマップと変数関連図に必要な情報を取得している．利用者がステップ実行させていくと同時に，メモリマップと変数関連図を作成しているので，確認したいソースコードの文を飛ばしてしまうと，再度初めからステップ実行でソースコードを辿らなければならない．可視化についても，変数関連図には変数名，または関数名のみの表示であるため，詳しい値などを知りたい場合にはメモリマップと交互に見比べる必要があり手間である．

文献 [3] で提案されているツールは，C 言語のソースコードとデータ構造の表を表示するだけでなく，プログラム構造を表すフローチャートを表示することでプログラム全体の流れを把握しやすい．また，ツール上でプログラム実行の開始・停止・一時停止・ステップ実行をすることができ，プログラムの一部の流れも把握しやすい．しかし，ソースコードを戻って辿ることはできない．文献 [3] では関数の可視化も行えるが，関数名のみで呼び出し関係の木を作図しているので，引数の関係など分からない．

文献 [4] では，実行トレースと実行時エラー検出を行う，C 言語初学者に焦点を当てたツール (SeeC) を拡張し，DOT 言語のスクリプトで表されたグラフを表現する Graphviz[5] を用いて，変数，配列，構造体，ポインタの状態を表したグラフを作成するツールを開発している．ソースコードと Graphviz の機能で作成した図のみ表示するので，各変数がどのような値を保持しているか分かりにくい．

既存のツールでは，利用者にステップ実行をさせ，変数関連図や変数の値をまとめた表などを随時表示している．また，変数などのデータを箱のように抽象化し，メモリアドレスを理解している必要がない可視化方法を採っている．しかし，間接演算子については触れられていることが少なく，初学者が意味を混同しやすいアドレス演算子と間接演算子についての違いを理解させるには向いていないと考える．本研究では，プログラム全体の動作を可視化し一覧性を持たせるとともに，利用者が選択したポインタ変数に関連する変数の関係図を作成し，プログラムの一部の動作も可視化する．これにより，利用者が確認したい箇所だけを表示できる．また，間接演算子についての表示を行うことで，アドレス演算子と間接演算子の違いを確認できる．本研究では，コンパイルの際に警告文が出るようなポインタを間違って使用している場合も想定し可視化しているので，利用者に対して間違いを気付かせられる．

3　動作理解支援方法の提案

3.1　支援対象の分析

3.1.1　ポインタについて

ポインタが理解できない初学者には，コンピュータの仕組みが分からずつまづく初学者と，C 言語のポインタの記述方法が分からない 2 通りの初学者がいると分析した．

コンピュータの仕組みが分からずつまづく初学者を生み出している原因は，ポインタの学習をするまで意識しなくても良かったメモリ空間とアドレスというものを学ばなけ

ればならなくなり，慣れていない概念に混乱しやすいからであると考えた．

ポインタの記述方法が分からない初学者を生み出している原因は，ポインタには混乱を招きやすい記述があるからであると考えた．ソースコード 1，2 にその記述例を示す．

ソースコード 1 は，ポインタ変数を宣言するとともに初期化をし，次の行で値を代入している．しかし，この記述を初学者が見た場合，2 行目で*p に対して&x というアドレスを代入しているにも関わらず，4 行目で同じ*p に整数値 5 を代入しており，*p にどの値が格納されているのか分からなくなってしてしまう可能性がある．また，ソースコード 2 のようなポインタ変数の宣言部分とは別の部分で初期化をしているような記述がある．ソースコード 2 の記述で，4 行目で p という変数に&x というアドレスを代入して，5 行目で*p に整数値 5 を代入している．初学者はプログラミング学習において，ソースコード 1，2 のどちらの初期化方法も教えられることで*と&の意味を混同してしまいがちである．

ソースコード 1　混乱を招くプログラム例 1

```
1  int x;
2  int *p = &x;
3
4  *p = 5;
```

ソースコード 2　混乱を招くプログラム例 2

```
1  int x;
2  int *p;
3
4  p = &x;
5  *p = 5;
```

ポインタを学ぶ上で，アドレスやメモリ空間といったものを理解することは重要である．しかし，メモリ空間などをイメージし理解するためには初学者が計算機の知識を身につけることが必要である．初学者が計算機のメモリの仕組みを理解していくには，バイトの概念など更なる知識を必要とするので，簡単に理解することは困難である．ここで初学者はポインタへの抵抗感を感じてしまいがちである．そこで，初学者が理解すべき初歩的な動作において，計算機の知識が必要ではないと考えた．例えば，最低限のポインタの動作理解をさせようとしたときに，初学者に与えるべき情報はポインタ変数が保持しているメモリアドレスと，ポインタ変数が指した先の中身の値の二つであると考えた．また，メモリ空間をそのまま可視化しようとした場合，ポインタを理解する上で必要のないデータも表示されるので見にくくなりやすいと考えた．

3.1.2　関数について

関数の学習において初学者が理解しづらい点は，実引数と仮引数の関係把握であると考えた．

実引数と仮引数の対応関係の中でも, ソースコード 3 のような場合が把握しづらいと考えた．

ソースコード 3　3 つの値を昇順に並べ替えるプログラム例

```
1   void swap(int *a, int *b){
2     int temp = *a;
3     *a = *b;
4     *b = temp;
5   }
6   void sort3(int *x, int *y, int *z){
7     if(*x > *y) swap(x, y);
8     if(*y > *z) swap(y, z);
9     if(*x > *y) swap(x, y);
10  }
11  int main(void){
12    int x, y, z;
13    x = 4;
14    y = 2;
15    z = 3;
16    sort3(&x, &y, &z)
17    return(0);
18  }
```

このときの sort3 関数の仮引数は，main 関数内の 3 つの変数各々のアドレスを保持しており，swap 関数の仮引数は，sort3 関数内での if 文の結果によって保持する変数のアドレスが変化する．このように条件によって参照する引数が変化をすると，初学者は関数の呼び出し関係について理解しづらいと考えた．また，初学者は main 関数内で呼び出されている sort3 関数の引数の記述部分を見て，“swap(&x, &y) ” のような記述をしてしまいがちであり混乱しやすいと考えた．

3.2　変数値履歴表と変数関連図による動作理解支援

3.2.1　提案する変数値履歴表

変数値履歴表を表示する際に，3.1.1 節で述べたように，最低限のポインタの動作を理解する上で初学者にメモリアドレスは数値としてではなくラベルとして捉えさせるべきであると考えたので，アドレスを&a，&b といった表記で示す．関数の表示は，関数が呼び出されるたびに関数の表のテーブルを生成し，関数内での値の変化を変数値履歴表に表示する．変数値履歴表の上部に所属関数名と型名を表示することによって変数の状態を確認しやすくした．

3.2.2　提案する変数関連図

2 節で述べたことを踏まえて間接演算子とアドレス演算子の意味を強調し，ポインタの学習におけるメモリ空間やアドレスの知識がなくとも，図を見ることで直感的に参照関係が分かるような可視化方法にする．

変数関連図の表示は，変数を表した四角いノードを図示し，そのノードの中に変数の値データを入れ，値の参照関係を表すためにノード同士を矢印でつなげるという可視化方法を考えた．また，ポインタ変数で間接演算子*がついたものに関しては，ポインタ変数はあくまでも参照先のアドレスを保持するということを強調させるために，図 1 のように間接演算子*が付いた変数名の表示を，ポインタ変数が指しているノードから吹き出しの表記を出して，その吹き出し中に変数名の表示をするとよいと考えた．また，

メモリアドレスを意識する必要をなくすことと，&演算子の意味を強調するためにポインタ変数の箱には&x，&y というように “&変数名” の表示にした．

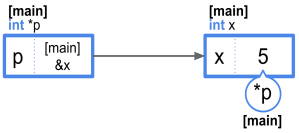


図 1　ポインタの可視化 (吹き出し案)

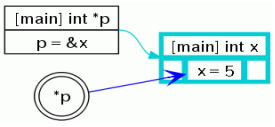


図 3　ポインタの可視化

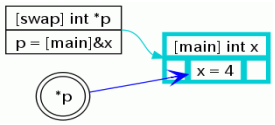


図 4　ポインタの可視化 (関数の引数)

4　ポインタの動作理解支援ツールの設計と実現

4.1　設計

本研究で提案するツールは，Java を用いて開発し，変数値履歴表の表示には Java の GUI ツールキットの Swing を利用する. 変数の値を表に表示し，ツール使用者に選択してもらったセルにある値の変数関連図を表示する．変数値などは GDB[6] の機能を使い取得する．また，変数関連図の作成については，ツールで DOT 言語のファイルを自動生成し，Graphviz[5] を用いて変数関連図を描画する．Graphviz は，DOT 言語のファイルを元に有向グラフの自動生成が行えることと，レイアウトを自動で整えて見やすくできる．

4.2　実現

ツールを起動すると図 2 のような画面が表示される．

先行研究 [1] で述べられているように表の列に行数，ソースコード，条件式の真偽，各変数の値が表示される．*付きの変数名，&付きの変数名のデータを取得できるようにすることで，ポインタ変数が保持しているアドレスを “&変数名” として表示することができた．また，図 2 のように，関数が呼び出されると新しい表テーブルを作成し，そのテーブルに呼び出された関数内の変数値の変化を可視化することで，学習者に関数の動作理解支援を行う．

変数関連図については，ツールの使用者に変数関連図の作成を行いたい変数のテーブル数，行数，列数の指定をしてもらい，ツールが dot ファイルの自動生成を行い，生成した dot ファイルを元に Graphviz が変数関連図の描画を行い，図 3 のような画像として出力が行われる．

図 3 は実際にツールを用いて可視化したものである．ツールを実装する上で，吹き出し表示ができなかったので，図 3 のように二重丸型のノードで表す代替案で実装をした．p と*p の違いを強調するためにエッジの色を変え，またエッジの終端をノードの外側と内側に指すようにした．

図 4 はソースコード 3 にツールを使用して生成された変数関連図である．関数の引数にポインタが使用されているときの可視化として，ポインタ変数が指している先の変数のアドレスと所属している関数を表示することで，学習者に引数が関数間でどのように渡されているのかをイメージさせる理解支援を行う．

5　考察

5.1　誤ったプログラムの可視化

初学者がエラー文や警告文を読むことで間違っているコードを修正することは難しい．ポインタが使われているプログラムに対して出る警告文は次のようなものが挙げられる．

1. ポインタ変数の初期化のし忘れ
2. 参照する変数の型が違う
3. 誤ったポインタへのポインタの使用

本ツールで以上のような誤ったプログラムに対しても可視化を行うことで理解支援できると考えた．

ポインタ変数の初期化のし忘れに対して，図 2 のような本ツールで生成された変数値履歴表で，ポインタ変数の値を確認し，&x，&y のような “&変数名” の表記になっていれば初期化できており，表記がメモリアドレス表記のままであれば初期化ができていないということを気付かせられると考えた．

参照する変数の型が違う場合，図 5 のような変数関連図が生成される．図 5 はソースコード 3 において，sort3 関数内の*a の型を double 型に書き換えた，*a についての変数関連図である．このときの変数関連図内の変数値の表示として，学習者が選択した変数の型に統一されるので，main 関数内で int 型で宣言されている変数 x の値が実数で表示されている．変数関連図内の型の表記と変数値の違いを見ることで，型の違いを気付かせることができると考えた．

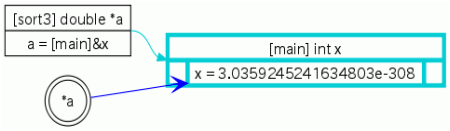
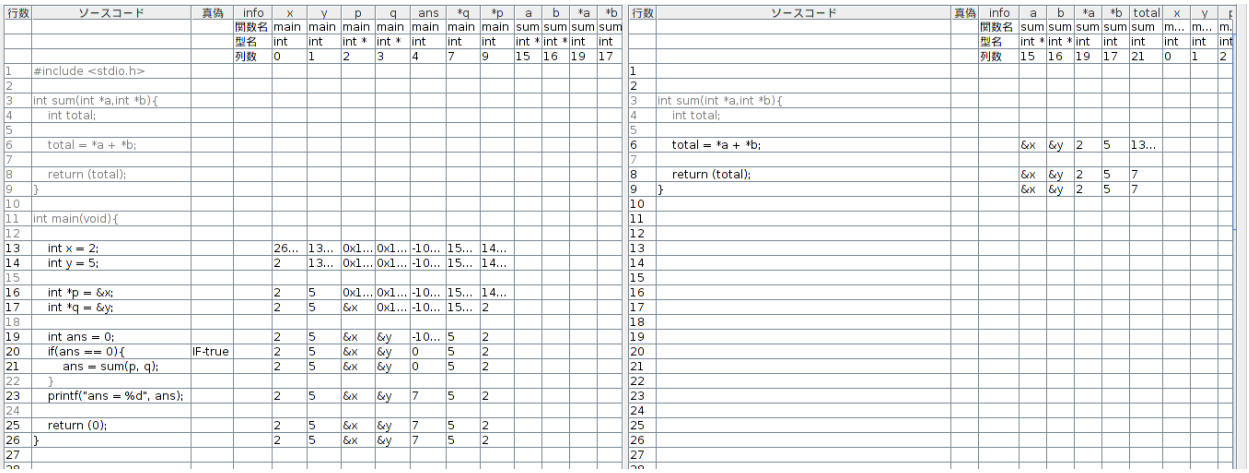


図 5　型の違いの可視化

本ツールでは，多重間接参照の表示を行うことができる．関数間での多重間接参照を図 6 のように可視化でき，この変数関連図を確認することで 3.1.2 節で述べたような誤った多重間接参照についても支援できると考えた．



行数	ソースコード	真偽	info	x	y	p	q	ans	*q	*p	a	b	*a	*b
			関数名	main	main	main	main	main	main	main	sum	sum	sum	sum
			型名	int	int	int *	int *	int	int	int	int *	int *	int	int
			列数	0	1	2	3	4	7	9	15	16	19	17
1	#include <stdio.h>													
2														
3	int sum(int *a,int *b){													
4	int total;													
5														
6	total = *a + *b;													
7														
8	return (total);													
9	}													
10														
11	int main(void){													
12														
13	int x = 2;			26...	13...	0x1...	0x1...	-10...	15...	14...				
14	int y = 5;			2	13...	0x1...	0x1...	-10...	15...	14...				
15														
16	int *p = &x;			2	5	0x1...	0x1...	-10...	15...	14...				
17	int *q = &y;			2	5	&x	0x1...	-10...	15...	2				
18														
19	int ans = 0;			2	5	&x	&y	-10...	5	2				
20	if(ans == 0){	IF-true		2	5	&x	&y	0	5	2				
21	ans = sum(p, q);			2	5	&x	&y	0	5	2				
22	}													
23	printf("ans = %d", ans);			2	5	&x	&y	7	5	2				
24														
25	return (0);			2	5	&x	&y	7	5	2				
26	}			2	5	&x	&y	7	5	2				
27														

行数	ソースコード	真偽	info	a	b	*a	*b	total	x	y	p
			関数名	sum	sum	sum	sum	sum	m...	m...	m...
			型名	int *	int *	int	int	int	int	int	int
			列数	15	16	19	17	21	0	1	2
1											
2											
3	int sum(int *a,int *b){										
4	int total;										
5											
6	total = *a + *b;			&x	&y	2	5	13...			
7											
8	return (total);			&x	&y	2	5	7			
9	}			&x	&y	2	5	7			
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											
27											

図 2　ツール画面

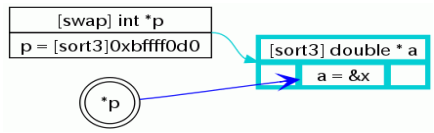


図 6　誤った多重間接参照の可視化

5.2　メモリ空間を意識させた可視化について

本研究では，プログラムの一部の状態を可視化し初学者に理解支援を行ってきたが，ある程度プログラミングを理解してきた学習者にはメモリ空間を意識させた可視化を行い理解を深めることができると考えた．メモリ空間を意識させた可視化において，関数内に存在する複数の局所変数がメモリ空間上では連続して格納されていることから，図 7 のように変数を関数ごとにグループ分けしたり，malloc 関数などを使いプログラム中で動的に確保されたメモリ領域は，別のグループに分けることで理解支援を行えると考えた．

初学者が変数の型と値を理解しやすいように本研究では図 3，4 のように，変数の型と値を分けて表示していたが，メモリ空間を意識した可視化では，メモリ空間全体の変数を表示する必要があるので見にくくなると考えた．そこで，図 7 のように変数の型と値の情報をまとめて表示することで，変数が多い場合でも見やすく表示できると考えた．

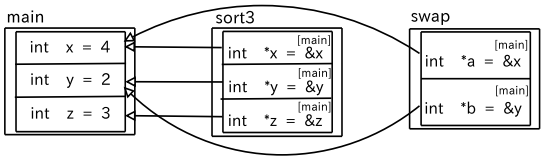


図 7　メモリ空間を意識した可視化

6　おわりに

本研究では，プログラミング学習の中でも初学者がつまづきやすい，ポインタと関数の動作理解を支援するための，動作理解支援ツールを提案した．GDB の機能を用いて変数やアドレス，関数の情報を取得し，グラフ作成ソフト Graphviz でポインタの対応関係を可視化した．ポインタ変数が指している先の値と，アドレスの区別ができるような表示にすることで，初学者に混乱を与えずに学習させることができると考えた．

今後の課題として，配列や構造体などの複数のデータを持つ構造の可視化や，コンパイル時に警告文が出るソースコードに対して，修正前の状態のプログラムの動作を可視化したものと，修正後の動作を予測し可視化したものを同時に学習者に見せ，プログラムの間違いを学習者に気付かせる機能の検討が挙げられる．

参考文献

[1] 長谷川洸也，川地周作，“命令型プログラミングにおける理解支援に関する研究”，南山大学情報理工学部 2014 年度卒業論文.

[2] 海老名慧士，“ポインタを使用したプログラムの可視化によるプログラミング学習支援”，芝浦工業大学ソフトウェア工学研究室，2009.

[3] 武田寛昭，山下英生，“C 言語プログラムに対するアルゴリズム可視化システム”，広島工業大学紀要研究編 第 42 巻， 2008， pp.247-253.

[4] Matthew Heinsen Egan，Chris McDonald，“Program visualization and explanation for novice C programmers”，ACE ’14 Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148 Pages 51-57

[5] 公式サイト『Graphviz - Graph Visualization Software』 http://www.graphviz.org/

[6] 公式サイト『GDB: The GNU Project Debugger』 http://www.gnu.org/software/gdb/