自己参照構造体を用いるプログラムの理解支援ツールの提案
ーリストを題材としてー

2016SE004 坂公博　2016SE065 奥村航
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データ構造について理解することは重要である．C 言語でデータ構造を構築する際に構造体とポインタを利用することがある．構造体のメンバに自身と同じ型のポインタをもつものを自己参照構造体という．自己参照構造体を利用することでリストのような基本的なデータ構造を実現できる．リストについて講義や参考書で学習する際，その構造を図示することが多い．学習者は参考書や講義で図示されたリストの構造を見ることで理解したつもりになるが，行う処理によってプログラムの実行状態がどのように変化するかわからず，リストに対する操作を実現するプログラムを作成しようとした際，コードが記述できない場合がある．

本研究ではリストの理解支援のために， C 言語を対象とするリストに対する操作を実現するプログラムの実行状態を可視化し，学習者の操作に対応する代入文を出力するツールを提案する．学習者が入力したリストのプログラムを逐次実行し，実行状態を可視化して図に表す．可視化した図から指し先を変更したいポインタと変更先を選択する．これらを基に図を変更し，その操作を行う代入文を提示する．学習者にツールを利用してもらうことで評価を行い，ツールの改善点と拡張について考察を行う．本研究では int 型，double 型をメンバとする一方向リストを繰り返しで処理するプログラムを対象とする．扱うプログラムはコンパイルでき，実行時エラーを起こさないものとする．

2　関連研究

文献 [1] は C 言語の関数の引数にポインタを用いたプログラムの状態を可視化する．誤りがある場合も可視化し，修正案を提示するが，構造体を利用するプログラムは対象としていない．

SeeC [2] では学習者の操作によってプログラムを逐次実行し，そのときの変数や関数の関係を図に表示している．SeeC では配列や構造体も扱うことができる．このツールではコードがある程度記述できていないと，可視化しても理解の助けにならない．本研究で対象とするコードが記述できない学習者への支援は難しい．

文献 [4] で提案されている VIE システムでは学習者の操作に応じて，コードの候補を提示するシステムを構築し，対話的に学習支援を行っている．可視化領域に作成されたオブジェクトに対して自由にマウスで操作することができ，それに応じたコードを提示している．定義できる関数は 1 つのみであり，扱える変数型は， int 型， char 型のみである．リストのように自己参照構造体を用いて，複数の関数で操作を実現するプログラムに対して支援を行うことができない．

3　理解支援方法の提案

3.1　リストにおける難しさの分析

学習者がリストに対する操作を実現しようとした際，コードが記述できない場合がある．その原因として，複数ステップで操作を実現する際，各ステップの実行状態をイメージすることができない，行いたい処理によって実行状態がどのように変化するかわからない，リストの要素をポインタで辿っていくなどのコードの書き方がわからない，ということがあげられる．

学習者はリストに対する操作を実現しようとした際，ソースコード 1 のような未完成の状態から，続きが記述できない場合がある．insert はリストに要素を挿入する関数であり， createList は空のリストを生成する関数である．

ソースコード 1　リストの n 番目と n + 1 番目の要素の入れ替えるプログラム (未完成)

```
1  struct node {
2      int val;
3      struct node *next;
4  };
5  typedef struct node Node;
6
7  void swap_nodes(Node *head, int n)
8  {
9      Node *prev = head;
10     Node *curr = NULL;
11     int i;
12     for (i = 0; i < n - 1; i++) {
13         prev = prev->next;
14     }
15     curr = prev->next;
16
17 }
18
19 int main(void)
20 {
21     Node *list;
22     list = createList();
23     insert(list, 0, 1);
24     insert(list, 1, 2);
25     insert(list, 2, 3);
26     insert(list, 3, 4);
27     swap_nodes(list, 2);
28     return 0;
29 }
```

コードを記述したとしても，処理の順番を間違い，図 1 のようになることがある．図 1 ではどこからも参照されない要素が生じてしまい，赤色のノードの入れ替えを行うことができない．赤色のノードの入れ替えを行うには図 2 のように処理を行う必要がある．

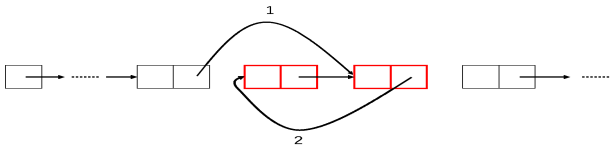

図 1　リストの要素の入れ替えを行う処理の間違った順番

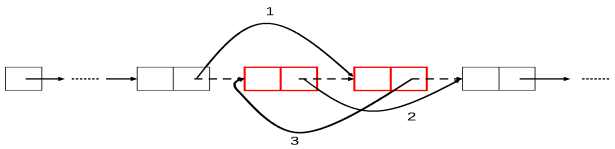

図 2　リストの要素の入れ替えを行う処理の適切な順番

3.2　理解支援方法

3.1 節でプログラムが記述できない原因としてあげた各ステップの実行状態がイメージできないことに対しては実行状態を可視化した図を提示することで支援を行う．学習者が行いたい処理によってプログラムの実行状態がどのように変化するかわからないこと，コードが記述できないことに対しては学習者の操作に応じて可視化の図を変更し，代入文を提示することで支援を行う．入力するプログラムで変数宣言以外に処理がされていない変数は初期化されていること，循環参照がないことを前提とする．

3.2.1　可視化する図

本研究では次のように可視化を行う．

- 変数を関数ごとにまとめて表示する
- ヒープ領域のものについては関数外に表示する
- int 型と double 型の変数と構造体のメンバは 1 つの箱で表す
- int 型と double 型の変数に値が代入されていない場合，-99 と表示する
- 構造体は箱でメンバすべてを囲む
- 関数名，変数名，構造体のメンバ名は箱の左に表示する
- ポインタを矢印で表す
- 値が代入されていないポインタと NULL ポインタは矢印の先端のみで表す

図 3 はソースコード 1 の 15 行目を可視化した図である．

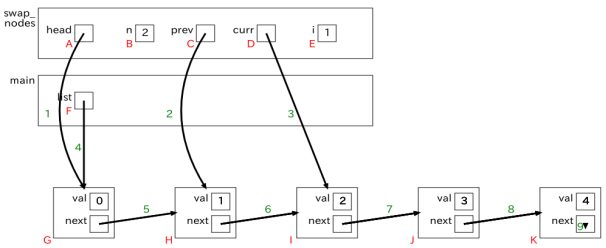

図 3　提案する可視化の図

3.2.2　図の変更・代入文の出力

本研究では可視化した図を操作することで代入文を提示する．代入文は，図 4 のように左辺の候補 = 右辺の候補と出力する．可視化した図に対する操作に対応するように可視化した図も図 5 のように変更する．

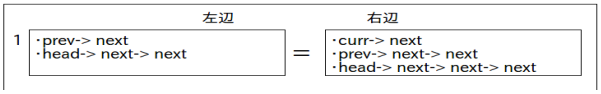

図 4　図 3 のポインタ 6 の指し先を J に変更した際の代入文

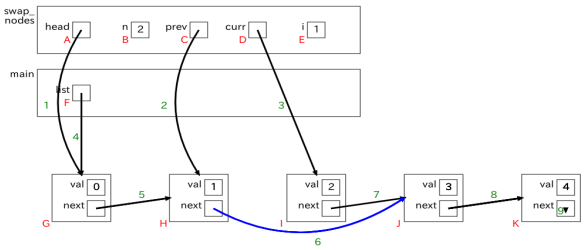

図 5　ポインタの指し先を変更した図

4　理解支援ツールの提案

4.1　ツールの概要

ツールを使用する際，ツールのダウンロードや端末でのコマンド操作を必要とすることは学習者にとって負担であると考えたので，提案するツールは Web 上で動作させる．学習者はツールの URL にアクセスすることでツールを使うことができる．学習者が C ソースファイルと実行プログラムへの標準入力をツールに送信すると，プログラムの可視化の図が表示される．学習者が可視化の図に対して，指し先を変更する操作をすると，図が変更され，代入文が提示される．学習者が行う図の操作は図に付けたラベルを選択する方法で行う．ラベルを選択する方法では間違った変更先を選択させないようにすることができる．描画できる範囲，ラベルの数の関係で関数呼び出しは 4 つまで，変数，ポインタの指し先は 26 個まで，データ型は int 型，double 型，構造体とそれらのポインタに限る．また，構造体を要素にもつ構造体，配列は扱わない．

4.2　設計・実装

ツールは学習者が入力した C ソースファイルと実行プログラムへの標準入力を受け取り，ツール内の CGI プログラムで C ソースファイルをコンパイルをして実行ファイルを生成し，標準入力部分はテキストファイルに出力する．実行ファイルと標準入力のテキストファイルからプログラムの各状態の情報を GDB [3] で取得する．取得した情報から図の描画を行い，可視化した図を表示する．指し先を変更するポインタとその変更先の情報を受け取ると，

その操作に応じた代入文が生成され，操作に応じて取得した情報を変更し，代入文と変更した図を表示する．文献 [1] を参考にして， GDB による情報の取得は Java で，図の描画は HTML の Canvas 要素と JavaScript を用いて実装した．

5 評価

5.1 実験

学部 3 年生 11 人を対象に実験を行った．被験者は以前，リストの要素の挿入，削除を行うプログラムを作成したことがあるが，入れ替えを行うプログラムは作成したことはない．被験者にはソースコード 1 の 15 行目より下を記述し，リストの要素の入れ替えを行う関数 swap_nodes を完成させる問題を課した．ツールを利用することでコードの書き方だけでなく，手順もわかるのか，手順がわかれば，ツールなしでもコードが記述できるのか，手順がわかる場合，ツールを利用すればコードが記述できるのかを検証するために次の 4 段階に分けて実験を行う．各段階で解けた被験者の想定される特徴は表 1 のようになると考えた．

1. リストの入れ替えを行う前の図と行った後の図を示し，問題に取り組む．制限時間は 10 分で行う．
2. ツールを利用して，問題に取り組む．制限時間は 10 分で行う．
3. ツールを利用せず，リストの入れ替えを行う手順を示し，問題に取り組む．制限時間は 5 分で行う．
4. 手順を示し，ツールを利用して，問題に取り組む．制限時間は 5 分で行う．

2 段階目で解けた被験者は 2 人， 3 段階目で解けた被験者は 3 人， 4 段階目で解けた被験者は 3 人， 解けなかった被験者は 3 人 であった．

表 1 各段階で解けた学習者の想定される特徴

段階	特徴
1	入れ替えの手順と処理に対応するコードの書き方について理解している．
2	ツールを利用することで，入れ替えの手順や処理に対応するコードの書き方が理解できた．
3	処理に対応するコードの書き方は理解していたが，入れ替えの手順がわからず，コードが記述できなかった．
4	入れ替えの手順は理解していたが，処理に対応するコードの書き方を理解していなかった．ツールを利用することで，コードの書き方が理解できた．

2 段階目で処理の順番がわからない場合でも， 1 段階目で記述したコードを直す被験者がいたのでツールを利用することで行いたい処理に対応するコードの書き方は学ぶことができると言える． 3 段階目で解けなかった被験者については処理の手順を示すだけでは実行状態の変化がわからず，行いたい処理に対応するコードも書けないと言える．解けなかった被験者の中にはツールで提示される複数の代入文をすべて記述した被験者がいたので複数の代入文を提示することで混乱してしまうことがあると言える．

5.2 リストに対する操作の実現

本研究で提案するツールを用いてリストの要素の挿入が実現できるかを確認した．リストの要素が 2 つある状態から 1 番目と 2 番目の要素の間に要素を挿入する操作を行う．プログラムの状態はソースコード 2 である．

ソースコード 2 リストの要素の挿入を行う関数 insert（未完成）

```
1  void insert(Node *head)
2  {
3       Node *p = head;
4       Node *c = NULL;
5       c = createNode();
6
7  }
```

変数 p は挿入する要素の前の要素を指す変数として操作をする．ツールでは新しく変数やリストの要素を作ることができないので，挿入する要素を malloc 関数でメモリ領域を確保し，変数 c で指すようにする．変数 p を挿入する要素の前の要素を指すようにする．要素の挿入を行うには，挿入する要素が挿入する要素の前の要素の指し先を指すようにし，前の要素が挿入する要素を指すようにすればよい．これらの操作を行った図は図 6 である．

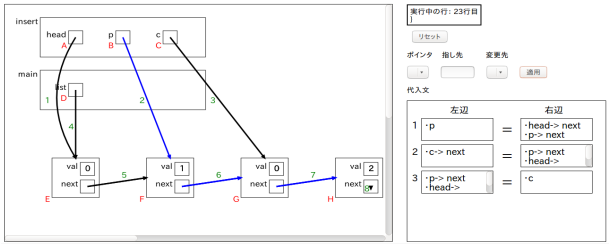

図 6 リストの要素の挿入を行った図

操作によって表示された代入文をプログラムに記述し，リストの状態を出力すると，実行結果 3 のようになり，要素の挿入が行われていることがわかる．

実行結果 3 2 番目に要素の挿入を行ったリストの状態

```
list: 1 0 2
```

以上より，新しく挿入する要素を malloc 関数でメモリ領域を確保されている状態のプログラムであれば，任意の場所に要素を挿入することができる．

同様にリストの要素の削除を実現できるか確認した．リストの最後の要素の削除を行う際，ポインタの指し先を NULL に変更することでリストの最後の要素の削除を実現することができる．

以上より，本研究で提案するツールでは，任意の要素の削除を学習者の操作によって実現することができる．

5.3 要素の挿入，削除の関数の一般化

実際の問題では， n 番目の次に要素を挿入するように一般化することが多い．n 番目の次に要素を挿入する関数 insert はソースコード 4 である．

ソースコード 4　リストの n 番目の次に要素を挿入する関数 insert

```
1  void insert(Node *head, int n)
2  {
3      int i;
4      Node *p = head;
5      Node *c;
6      c = createNode();
7      for(i = 0; i < n; i++){
8          p = p->next;
9      }
10     c->next = p->next;
11     p->next = c;
12 }
```

5.2 節で行った操作で出力された代入文とソースコード 4 を比べると，2 ， 3 番目の操作で出力された代入文は同じで，変数 p を挿入する要素の前の要素を指すようにする操作は for 文で記述されている．変数 p を挿入する要素の前の要素を指すようにする操作は挿入する場所によって変わる． 1 番目の次に挿入するときは p = p-> next ， 2 番目に挿入するときは p = p-> next-> next となる． 1 番目のときは 1 回， 2 番目のときは 2 回 p = p-> next を実行しているので， n 番目の次に要素を挿入するときは n 回 p = p-> next を実行すればよいとわかればソースコード 4 のように for 文を使って書き換えられると考えた．

6　考察

6.1　可視化の図

操作によっては操作前の図と操作後の図が大きく変わってしまい，見づらくなる場合がある．例えば，図 3 に対してポインタ 2 の指し先を J に変更する操作をした図 7 があげられる．

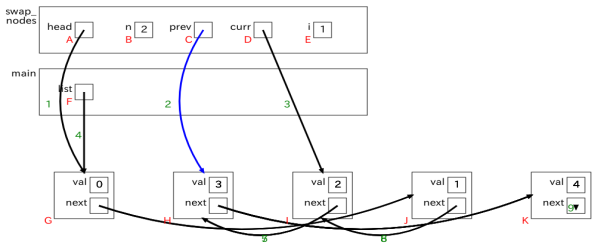

図 7　図 3 のポインタ 2 の指し先を J に変更する操作

情報の取得，更新をした後に，ヒープ領域のものをアドレス順にソートしてから描画することで操作をする前と後で図が変わらないようにできるのではないかと考えた．

6.2　再帰関数

再帰関数を用いてリストを実装したプログラムでツールを利用したとき，同じ関数を複数回呼び出しており，すべて描画できない．出力する代入文は呼び出している関数から辿れるもののみ表示するようにしているので，同じ関数を複数呼び出す再帰では代入文の出力が行えない．

6.3　データ構造の拡張

6.3.1　双方向リスト

双方向リストでは，循環参照があるので， GDB でプログラムの情報を取得するときに無限に情報を取り続けてしまう．双方向リストに対応させるには，情報を取得するときに値，アドレスの両方が同じポインタをその実行状態のポインタの数以上取得したら情報の取得を終了するようにすればよいと考えた．

6.3.2　木構造

木構造を実現したプログラムでツールを利用したとき，ポインタの指し先を変更する操作と代入文の出力は問題なく行われるが，可視化の図で木の要素が一直線に表示されるので，見づらくなる．

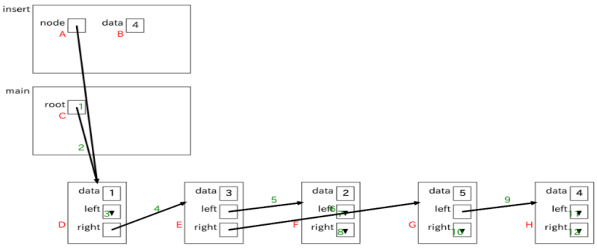

図 8　木構造を可視化した図

7　おわりに

本研究では C 言語を対象とし，リストを題材とした自己参照構造体を用いたプログラムの理解支援ツールの提案をした．プログラムを逐次実行し，実行状態を図示する．学習者の操作に対応するように図を変更し，代入文を出力する．これらによって，視覚的にプログラムの実行状態を把握することができ，学習者に自分の操作に対応する代入文の書き方を理解させることができる．今後の課題として可視化の図の改良，一方向リスト以外のデータ構造への対応，free 関数への対応があげられる．

参考文献

[1] 加藤ちひろ，松尾翔馬，森本朱音：関数の引数にポインタを用いたプログラムの可視化による動作理解支援ツールの提案，南山大学理工学部 2018 年度卒業論文 (2018).

[2] SeeC - program visualization and debugging for novice C programmers, available from <https://seec-team.github.io/seec/index.html>(accessed 2020-01-13).

[3] The GNU Project : GDB: The GNU Project Debugger(online), available from <https://www.gnu.org/software/gdb/>(accessed 2020-01-13).

[4] 川崎雄登，平井佑樹，金子敬一：プログラミング学習のための可視化対話環境，情報教育シンポジウム 2014 論文集，Vol.2014，No.2，pp.143-150(2014).