Science Journal of Kanagawa University **33** ：13-17 (2022)

■原　著■

C 言語のポインタ教育で用いるコードと説明図の同時作成を補助するシステム

栗原優太 [1]　永松礼夫 [1, 2]

A support System for Co-Creation of Code and Explanatory Figure Used in Education on C Programming Language Pointers.

Yuta Kurihara[1] and Leo Nagamatsu[1, 2]

[1] Department of Computer Science, Faculty of Informatics, Kanagawa University, Yokohama City, Kanagawa 221-8686, Japan
[2] To whom correspondence should be addressed, E-mail: lnag@kanagawa-u.ac.jp

Abstract: A support system was developed to assist in the preparation of lecture handouts and assignment materials. This system is able to co-generate a target program source code and an explanatory figure used in the teaching of C language pointers. Targeted class activities are "write explanatory figure from the code" and "write code from the explanatory figure". We developed the system for some typical exercises – follow the steps of a linear list operation and draw a figure that represents the structure of the list. The system consists of two parts – a generator and a viewer. The generator forms connection patterns between structures, C language codes, and data for displaying an explanatory figure based on specified settings. The viewer displays/edits/saves the explanatory figure of data structures. The viewer includes simple editing and saving features. As a result of actual use of the system, issues such as the lack of quality control of connection patterns that can be generated and the lack of an explanatory figure of the steps in the code emerged. We are able to effectively create code and an explanatory figure useful in the exercises.

Keywords: C programming language, programming education, pointer, linear list, source code generation, explanatory figures

序論

本学科を始めとする情報 /CS 系の学部では、コンピュータの動作原理や基幹技術の知識体系を深く学習するために、高水準言語の中でハードウェアに近い機能を扱うことが可能な C 言語の学習をカリキュラムに取り入れることが多い。C 言語はポインタやメモリ管理の部分が難しく、挫折する学生の割合が高い。

先行研究としては、これを解決するための試みとして、金子らが開発した VIE システム [1, 2]、小池らが開発した SuZMe[3] がある。両者ともコードを学習者がステップごとに実行して、ポインタの関係を図から理解する補助システムである。VIE システムは変数のブロック構造 (型・変数名・値) の可視化、SuZME はメモリ空間 (値とアドレス) を可視化という機能も備えている。

これらのシステムでの可視化とは、作成済みプログラムの動作のポインタで指し示す関係の変数を表す長方形とそれらを結ぶ矢印として図示を行うもので、主なユーザは学習者（学生）を想定する。ポインタの教育では教員が授業資料などを手早く作成することも大事であり、対象とするソースコードと解説図をペアで生成したいと考えた。

本研究では、主たるユーザを課題作成者（教師）とし、教材や課題に使用できるコードや図の作成補助を行う課題作成補助システムの開発を目指した。

目標

ポインタをわかりやすく教える方法として、コード

とメモリの状態の解説図を関連付けて理解してもらう方法がある。それを行うため「コードから解説図を書く」「解説図からコードを書く」という演習を実施することが想定される。

これらを支援するための「教員が演習問題の設定を入力すると、それのコードと解説図の生成を行うシステム」の開発を目標とした。また、「事前の説明に使用する例題」や「学生に解いてもらう演習問題」、「難易度を変更した類似する演習問題」といった、複数の類似する問題の作成を可能にすることも目標とした。参考とした例題[4]の形式をベースとし、主に線形リストを対象にするシステムとした。

先行研究のシステムとの違いを、表1に整理した。構造体や自己参照構造体（構造体へのポインタ）について、先行研究では論文内に記載がないため未対応と推定している。

表1. 先行研究のシステムとの違い

	VIE	SuZME	本研究
ポインタ変数の指す先を矢印で繋ぐような形式での、コードの図示	○	○	○
メモリ空間をアドレスと値の表として図示	×	○	×
教材に使用する新規コードの生成	×	×	○
システム単体での、コードの実行とデバッグ	○	×	×
コードの実行ステップ毎の状態の図示	○	○	×
図示された図を操作すると、対応するコードの提案を行う	○	×	×
構造体や自己参照構造体を使うコードの図示	?	?	○
「コードから図」「図からコード」という例題への対応	○	○	○

方法

ベースとする演習の形式[4]は、本学科の2年生を対象とした授業内演習のものである。実際の授業では以下の流れで実施していた。

最初に教師は、自己参照的ポインタと構造体で線形リストを表現するコード、具体的には値とポインタの2つのメンバを持つ構造体が複数あるものについて、コードとリストの構造を表す解説図を掲載した講義資料(サンプルは図1と図2)を作成する。コードはステップ毎にラベルが付いている。解説図は、構造体を表す箱と、ポインタのリンクを表す矢印で構成されている。矢印はコードの処理が進むと生成・削除される過程をステップの記号を付記して示してある。

```
int main(void){  //以下の代入ステップを追う
   t = mk_item(4,&n3);//※1 --a
     p1=(struct item*)malloc(sizeof(struct item)); //a1
     p1->num=4; //a2
     p1->nxt=&n3; //a3
     t=p1; //a4  4ステップに分解
   s = &n2; //--b
   n3.nxt = s; //--c
   s->nxt = &n1; //--d
   s->num = 5; //--e
   s = mk_item(6,t); //※2 --f
     p2=(struct item*)malloc(sizeof(struct item));
     p2->num=6; //f2
     p2->nxt=t; //f3
     s=p2; //f4  4ステップに分解
   for(  ; s!=NULL; s=s->nxt)
       printf("%d¥n",s->num);
```

図1. 対象とする演習で用いるC言語コード.

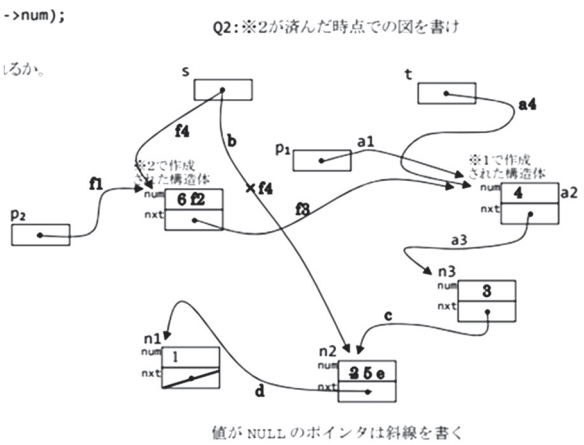

図2. 対象とする演習で用いるリスト構造の図.

この資料を印刷して学生に配布し、教師はコード実行中のリンクの生成・削除や値の変更の動きを、話しながら赤ペンでなぞるなどしてステップごとの動作を解説する。その後、学生に別のコードと構造体を表す箱だけの解説図の雛形が書いてある用紙を配布し、約10分間でポインタのリンク（矢印）と対応するステップの番号を書きこむ演習を行った。

システムの機能と生成物の要件

先述したような形式の演習問題の設定（構造体のメンバ・宣言する構造体の変数等）を入力すると、コードとそれが表現する線形リストの解説図を生成するようなシステムの開発を行った。

コードはステップ毎に値の書き換えを行うものとする。類似問題の量産を行うために、ポインタの指

す先のランダム変更ができるようにする。実行過程の解説用に、コードのステップ毎にコメントで番号を振るようにする。構造体は静的確保（事前に宣言）と動的確保(malloc で確保)から選択できるようにし、それぞれの違いの教育にも使用できるようにする。解説図で構造体変数を表す箱は、図 2 のように上から順番にメンバが並ぶような形とする。

システムの構成と開発環境

システムは、2 つの GUI アプリケーションで構成する方針とした。一つ目は、構造体間の繋がりパターンと C 言語コード・解説図表示用データを生成 / 保存を行う Windows 専用デスクトップアプリ（以下、ジェネレータ）であり、二つ目はデータ構造の解説図の表示 / 編集を行うウェブアプリ（以下、ビューワ）である。

システムの設計：各種データのジェネレータ

ジェネレータには、先述したような形式の課題に使用するコードと解説図を作成する上で手間となる、①構造体間の繋がりパターン②データ構造の解説図表示用データ (JSON)、③線形リストを表現するコードの 3 つを生成する役割を持たせる。

操作は、「保存済み設定のインポート（任意）」、パート A「使用する構造体の設定決定」、パート B「構造体変数の設定と、繋がりの生成」、パート C「解説図表示用データと C 言語コードの生成、それらと構造体間の繋がりパターン設定の保存」の順番で行う。各パートの具体的な仕様について述べる。パート A「使用する構造体の設定決定」（図 3）では、使用する構造体の設定を決める。今回は一つの課題で使用できる構造体の種類は一個のみとする。使用できるメンバの型は「int, float, double, char, char（文字列）」および同じ構造体へのポインタとする。メンバのポインタの数は複数個設定できるようにした。コードのプレビュー機能により、意図している構造体になったかどうかを確認できるようにする。

パート B「構造体変数の設定と、繋がりの生成」（図 4）では宣言する構造体変数を一つずつ入力し、それらの間の繋がりのパターンの生成を行う。生成は現時刻をシード値とする乱数で行う。左側のパネルに宣言する構造体変数の名前とメンバの初期値（必須）を入力する。構造体変数は「実体、構造体へのポインタ、新規作成した構造体へのポインタ」の三種類の中から選択できるようにする。ポインタであるメンバのうち、他の構造体に繋げたいポインタは NONE と入力する（後の処理でランダムに選択された他の構造体へのポインタまたは NULL ポインタが割り当てられる）。「新規作成した構造体へのポインタ」は、malloc で動的確保した構造体と、それへのポインタのセットのことである。繋がりのパターンは、ループする繋がりを含むか否かを切り替えることができるようにした。右側パネルの「繋がりをランダム生成」ボタンを押すと、構造体変数とメンバの値のリスト（繋げる前と繋げた後）をそれぞれ生成して表示する。動的確保した構造体は【nn】、それへのポインタは【np】と右側に表記する。

パート C「解説図表示用データと C 言語コードの生成、それらと設定の保存」（図 5）ではパート B で作成した繋がりのパターンに基づき、解説図表示用

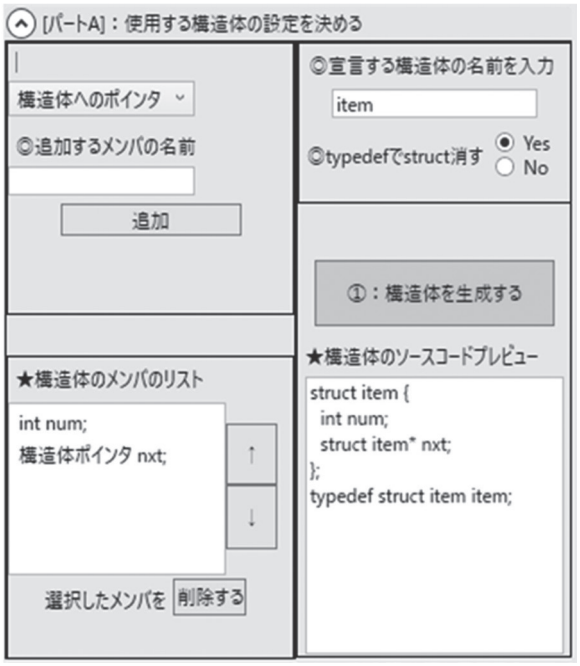

図 3. パート A「使用する構造体の設定決定」.

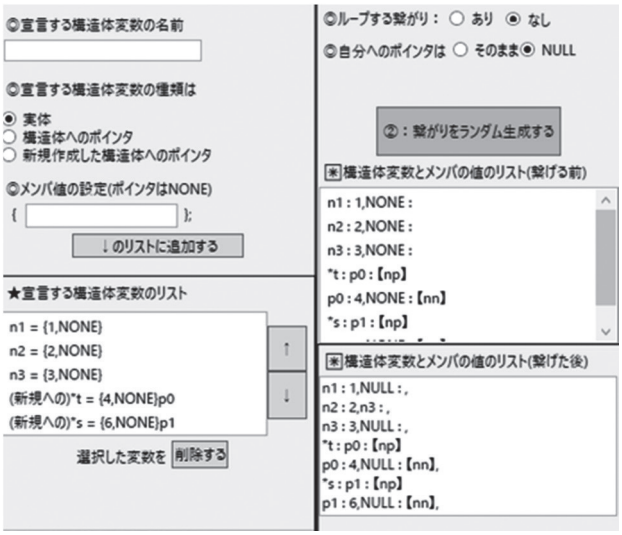

図 4. パート B「構造体変数の設定と繋がりの生成」.

図 5. パート C「解説図表示用データと C 言語コードの生成, 構造体間の繋がりパターン設定の保存」.

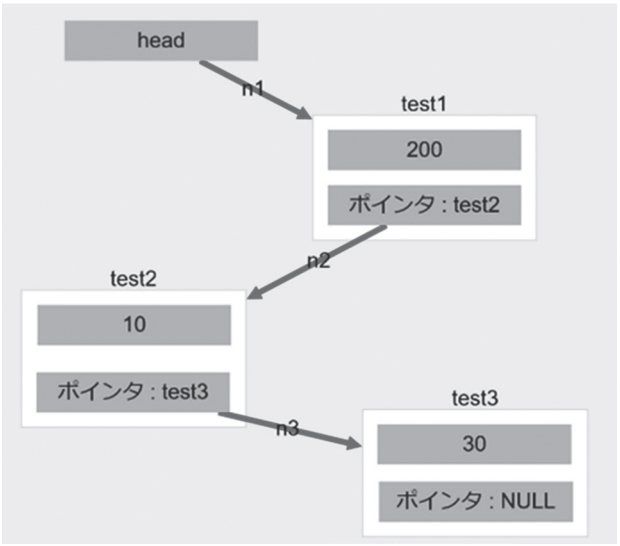

図 6. 最も簡単なパターンでの解説図の例.

データと C 言語コードを生成する。名前を入力し保存ボタンを押すと、保存用フォルダにそれらと、インポート用の設定を保存する。解説図表示データは「.graph.json」、設定は「config.json」の拡張子のファイルに保存する仕様とする。解説図表示データに関して、迅速な開発を行うため、今回の版ではノードの座標情報を含まない仕様とする。

「保存済み設定のインポート（任意）」について、ここではパート C で保存した「config.json」を、インポート部分で読み込み、構造体 / 変数の設定が再利用できるようにする。

システムの設計：解説図のビューワ

ビューワには、各種データのジェネレータで生成された解説図表示用データ「.graph.json」から、各構造体とそれらのメンバ変数の値を示す解説図を表示する役割を持たせる。

構造体とメンバは四角形の箱型ノードで表現する。ポインタはメンバから構造体に伸びる矢印で表現する。また、簡易的な編集機能を持たせ、ノードや矢印の削除、矢印の付け直しをできるようにする。

図の表示には JavaScript のネットワーク図描画ライブラリの Cytoscape.js[5] と、それの機能の拡張を行うライブラリ群 [6] を使用する。ノードをグループでまとめる機能を持っていることから、線形リストの解説図を表現することに適していると考え、採用した。今回の版では、迅速な開発を行うために、ビューワ上で編集可能な要素は図のみとする。そのため、ビューワの時点でコードを修正したくなった場合は、ジュネレータに戻って生成をやり直すことになる。また、今回はジェネレータの項で述べたように解説図表示用データ「.graph.json」には、ノードの座標情報を含まないため、読みこみを行った時点では、全ての構造体のノードは横並びに配置されている。そのため手動でノードを移動させる編集を行って整える操作が必要である。移動後の解説図の例を図 6 に示す。

結果

「システムの機能と生成物の要件」の項で定義したようなシステムを開発することができた。生成物のサンプルは図 7 と図 8 である。ステップ毎に番号があるため、実行過程の解説を行うことができる。

類似問題の量産について、構造体間の繋がりを乱数でランダムに決定する形式にしたため、教材として適切なパターンをうまく生成できなかった。具体的には、ポインタの指す先が 1 つの構造体に集中せず分散しているまたはポインタでの繋がりから分断された孤立している構造体が無いようなパターンが望ましい。また、繋がりがノード A から始まり、ノード B で終了するといった特定のノードに着目したようなパターンを選んで生成したい。しかしランダム生成の結果からこういったパターンが出てくるまで

```
int main(void) {
  n2.nxt = &n3;  // f1
  n1.nxt = &n2;  // f2
  n3.nxt = NULL;

  t = make_node(4, &n2);
  // p = (item *)malloc(sizeof(item));
  // p->num = num;
  // p->nxt = nxt; //f4
  // t =p; //f3

  s = make_node(6, &n2);  //
  // p = (item *)malloc(sizeof(item));
  // p->num = num;
  // p->nxt = nxt;// f6
  // s =p; //f5
}
```

図 7. 例題に近い設定で生成したコード.

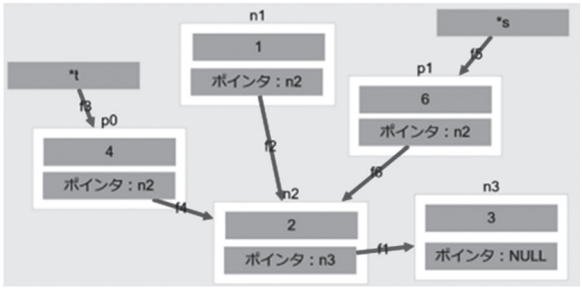

図 8. 例題に近い設定で生成した解説図.

何度も生成をしたり手動で少し手直しを行ったりする手間は煩わしい。

システムの使用で判明した知見

想定していたような形式の演習に使用できるような、「コードとデータ構造の解説図」を作成することに成功した。ただし、先述した通り、構造体間の繋がりのパターンの生成方法がランダムであるため、質は不十分であった。作成した 1 組の「コードとデータ構造の解説図」を調整したものに問題文を付けて、演習資料を作成する作業は、解説図の横並びになっているノードを適切な位置に移動させる時間も含め約 20 分で完了した。また、保存済みの設定の中から「使用する構造体変数の個数や繋がりの複雑さ（繋がりの個数とループする繋がりの有無）」といったデータ構造の複雑さに関する項目を変更し、データ構造理解の難易度を調整した課題を複数作成することに成功した。

解説資料を作成する上で、途中のステップ時点での解説図を作れないことが問題点として浮上した。コードの処理が完了した時点の解説図のみを生成する仕様だが、これでは途中経過の説明を行うことが難しい。また、コードにポインタ以外のメンバの値の書き換えを行う処理を追加しないと、作成できる課題がやや単純なものになってしまうことも判明した。

討論

システムの使用により必要だと判明した「構造体間の繋がりのパターンを、人間が手動で作成した課題に近いものにする機能」、「途中のステップ時点での解説図を作成する機能」、「型がポインタ以外のメンバの値の書き換えを行う機能」の追加実装が課題である。途中のステップ時点での解説図は、コードのステップ毎に、ノードと繋がりを追加した解説図を生成するような、単純な仕組みの導入で容易に実装可能である。

繋がりのパターンを、システムの開発結果で述べたような人間が作成した課題に近いものにするためには、それらの規則性を再現可能なアルゴリズムを作成し、それを用いて適切な行き先の選択を行うことが有効だと考えられる。ポインタ以外のメンバの値の書き換えは、書き換え対象と値を選択する操作ボタンや処理の追加が必要である。

線形リストを扱う課題だけではなく、二分探索木を始めとするより幅広いアルゴリズムやデータ構造に対応することも求められる。現時点では、構造体のメンバにポインタ変数が 2 つ以上ある場合にも対応しているため、通常の二分木なら作成できる。しかし、ノードの値の大小判定をする機能がないため、二分探索木への対応はできない。このような、各アルゴリズムに沿った生成処理を行う機能の追加とそれを制御する UI のデザインを検討する必要がある。

本研究のシステムのジェネレータは、生成中は繋がりの状態を一対一の接続関係の列挙で管理する形式である。そのため、先述したようなパターンやデータ構造のアルゴリズムと関連させて繋がりの状態を扱うことが困難である。今後は、繋がりの関連性を容易に取得・操作できるような状態管理の導入が求められる。

結論

「教員が演習問題の設定を入力すると、それのコードと解説図の生成を行うシステム」の開発を行った。それを用いて「授業の事前説明に使用する例題」や「学生に解いてもらう演習問題」、「難易度を変更した類似する演習問題」の作成ができることを確認した。

文献

1) 吉田英樹，金子敬一 (2009) プログラミングにおけるポインタ操作を通した関数学習機能．情報処理学会全国大会講演論文集第 71 回 コンピュータと人間社会：p587-588.
2) 川﨑雄登，平井佑樹，金子敬一 (2014) プログラミング学習のための可視化対話環境．*情報処理学会 情報教育シンポジウム 2014 論文集* 2 号：p143–150.
3) 小池伸弥，郷健太郎 (2012) プログラム実行時のメモリ空間可視化におけるインタラクションの提案．*情報処理学会シンポジウム論文集* 3 号 論文番号 :2EXB-36.
4) 2013 年度・神奈川大学・情報科学科・授業資料，プログラミング 2・第 13 回授業内演習.
5) Max Franz, Christian T. Lopes, Gerardo Huck, Yue Dong, Onur Sumer and Gary D. Bader(2016) Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics* Volume 32 Issue **2**: p309-311.
6) U. Dogrusoz, A. Karacelik, I. Safarli, H. Balci, L. Dervishi and M.C. Siper(2018) Efficient methods and readily customizable libraries for managing complexity of large networks. *PLoS ONE* **13**(5): e0197238.