N-011

制御構造グラフを用いたソースコード理解のための演習支援手法
Exercise Supporting Method for Understanding of Source Code Using Control Structure Graph

石原 俊 † Shun Ishihara　田口 浩 † Hiroshi Taguchi　原田 史子 ‡ Fumiko Harada　島川 博光 ‡ Hiromitsu Shimakawa

1. はじめに

C 言語プログラミング教育において，複雑な機能を実現するソースコードの読解は重要な学習項目である．プログラミング能力育成過程の中で学生は，複雑なソースコードを読解し，コードが表現するアルゴリズムを理解する能力を身につけなければならない．我々は，学生の到達度を検証するため，自動採点システム [1] を用いたプログラミング実技試験を実施した．その結果，読解演習に用いられたコードと同じコードを書かせる試験問題であったにも関わらず，全受検者の半数近くが無得点であった．これはソースコードに対する理解度を小さな人的負荷で判定することの難しさから，多くの学生が読解演習を低い到達度のまま通過していることに起因すると考える．

本論文ではソースコードに対する理解度判定のための一手段として，機能拡張などの仕様変更をソースコードに反映させる演習および，支援手法を提案する．ソースコードの直接的な修正は理解度の高い学生にしか実行できない．そこで支援手法は，ソースコードより制御構造グラフを生成し，これを解答作成環境として学生に提供する．これにより，ソースコードを直接修正させる場合に比べて，さまざまな理解度を持つ学生の各々に対して理解度の判定が可能になる．

2. 大学におけるプログラミング教育

2.1 2 段階の教育

プログラミング教育には二つの段階が存在する．第一段階は，学生にプログラミング言語の文法などの基礎知識を定着させる段階である．第一段階における有効な演習方法は，教員から新たに単元を教わるたびに，教わった知識を適用すれば容易に解けるようなプログラミング課題を複数課すことである．

第二段階は，基礎知識を応用し，実際の複雑なプログラム仕様からプログラムを生成する能力を育成する段階である．第二段階における有効な演習方法は，ソートアルゴリズムなど複雑なプログラム仕様を学生に理解させたうえで，仕様から実装されたソースコードを読解させることである．学生は高度なソースコードを読解することで，そこから基礎知識の実問題への応用方法を学び取る．しかしながら，ソースコードに対する理解度を小さな人的負荷で判定することは難しい．そのため，多くの学生が不十分な到達度にも関わらず読解演習を通過してしまい，基礎知識の応用力を身に着けられていない．

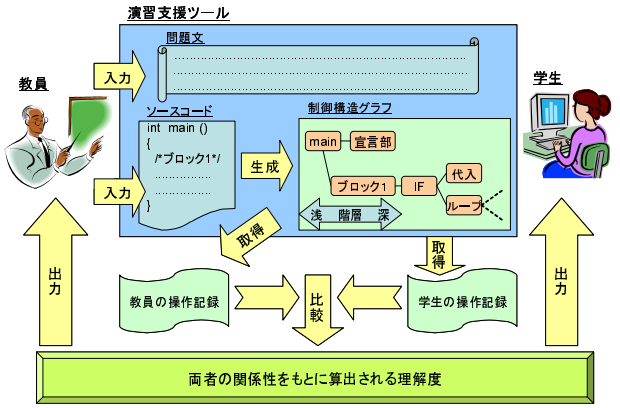

図 1　演習の流れ

2.2 ソースコード理解のための演習

ソースコードに対する理解度判定のための一手段として，仕様の変更をソースコードに反映させる演習を行い，反映後のソースコードを評価する方法が考えられる．演習の例として，単方向リストの操作を実装したソースコードに対し，双方向リストを扱えるよう機能拡張させるような演習が挙げられる．しかしながらソースコードの修正は理解度の高い学生にしか不可能であり，さまざまな理解度を持つ学生の各々に対して適切な評価を与えることは難しい．この演習によって理解度を詳細に把握するには，ソースコードを直接修正するような具体的な解答方法ではなく，より抽象的な解答方法を学生に提供する必要がある．

3. 制御構造グラフを用いた理解度判定

3.1 演習の全体像

本論文では，仕様の変更をソースコードへ反映させる演習を対象に，ソースコード修正に比べてより抽象度の高い解答方法を提供する手法を提案する．提案手法を取り入れた演習の流れを図 1 に示す．

演習準備として，まず教員は演習に必要な問題文とソースコードを演習支援ツールに入力する．ソースコードには，機能ブロックの区切りが特殊なコメント記号によって追記されている．教員の入力を受けると，ツールはソースコードから制御構造グラフを生成する．制御構造グラフとは，ソースコードを階層的に単純・抽象化し，制御構造や機能ブロックを視覚的に分かり易くグラフ化したものである．グラフには，機能ブロック，制御構造，具体的処

† 立命館大学大学院理工学研究科
‡ 立命館大学情報理工学部

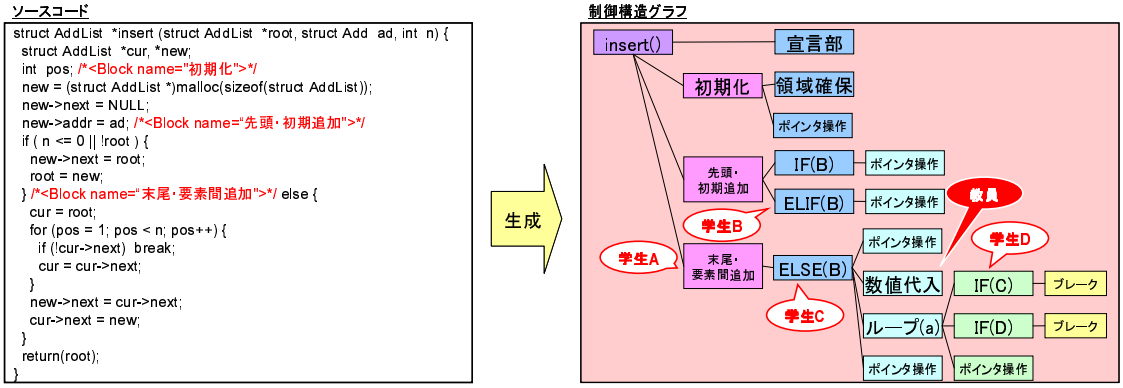

図 2　コードから生成される制御構造グラフと学生による解答方法

理のように，抽象度の高い順に階層が存在する．

次に教員は，問題文に書かれた機能を新規追加するさいに制御構造グラフ中で変更の必要になるノードをすべて指定する．そのさい，教員は可能な限り下位階層のノードを指定する．ツールは指定されたすべてのノードを記録する．この状態で教員は学生にツールを渡し，演習を実施する．学生も教員と同様に，変更の必要になるノードを可能な限り下位で指定する．

最後に，ツールは教員と学生双方の指定操作記録を比較し，両者の一致度を算出する．両者が完全に一致していれば最高の理解度であると判断する．両者が一致していなければ，両者の木構造上での関係を調べる．両者が親子関係にあればある程度の理解度に到達している，そうでなければ低い理解度であると判断する．このように制御構造グラフを演習に用いることで，幅広い理解度の判定が可能になる．

3.2　コードからの制御構造グラフ生成

提案手法はソースコードから制御構造グラフを生成するコンパイラを提供する．グラフは XML など階層構造を表現可能な言語で規定される．ソースコードから生成される制御構造グラフと学生による解答方法を図 2 に示す．まず，コンパイラは機能ブロックを最上位要素として抽出する．図 2 において **Block** とコメントの入っている位置から，再び **Block** が現れるまでのコードが 1 つの機能ブロックである．

次に，すべての機能ブロックに含まれるコードをノード化し，属する機能ブロックに子として接続する．そのさい，switch や for などの制御構造はすべて IF-ELIF-ELSE，ループ，ブレークの 3 種類の要素で再構築され，1 つのノードとして接続される．接続処理を再帰的に行っていくことで，多階層に及ぶ制御構造グラフが構成される．

3.3　学生による解答方法と解答の評価方法

学生は制御構造グラフから，問題文に記されている仕様変更に際して修正が必要になるノードを指定する．ツールは事前に教員より指定されたノード (以下 T) と，学生により指定されたノード (以下 S) との関係を調べる．T と S との関係は以下の 3 ケースに分類される．

ケース 1：T と S が同じノードである
ケース 2：S が T の親ノードとなっている
ケース 3：ケース 1，2 以外の関係である

ケース 1 の場合，学生のソースコードに対する理解度は最高水準にあると判断される．ケース 2 とは，図 2 における学生 A と教員，学生 C と教員のような関係である．ケース 2 の場合，ケース 1 より学生の理解度は劣るが，ある程度の水準に到達していると考えられるため，システムは教員のノードにより近い親関係にあるノードを指定できている学生ほど高評価を与える．図 2 の例では学生 A より学生 C の方が高い評価となる．学生 B，D はケース 3 に分類され，最低水準の理解度にあると判断される．

4.　既存手法との比較

抽象的な解答方法を学生に提供する演習支援手法に ProGuide [2] がある．まず，2.1 で論じた，手法の対象とする教育段階について比較する．本手法が第二段階を対象としているのに対し，ProGuide は第一段階を対象とした手法である．

次に，支援可能な演習課題について比較する．ProGuide は第一段階を対象としているため，現在 2 つの演習問題しか支援できない．それに対し本手法は，既存ソースコードへの仕様変更という形式であれば，どのような問題でも支援できる．

5.　おわりに

本論文では，仕様の変更をソースコードへ反映させる演習を対象に，抽象度の高い解答方法を提供する手法を提案した．今後の予定として，プロトタイプツールの実装，大規模講義への導入による有効性の検証などを考えている．

参考文献

[1] 石原 俊, 田口 浩, 高田 秀志, 島川 博光: ”マークアップによる C 言語プログラミング試験採点システム”, DEWS2007, D9-1 (2007)

[2] Cristiana Areias, Antonio Mendes: ”A tool to help students to develop programming skills”, CompSysTech'07, pp.(IV.20-)1-7 (2007)