Vol.2015-CE-132 No.13
Vol.2015-CLE-17 No.13
2015/12/5

自習に適したプログラミング教育用例題自動生成システム

若谷 彰良[1,a)]　前田 利之[2]

概要：プログラミング教育においては文法のような知識習得だけでなく, 実際にプログラムを作成する体験がその理解には重要となる. 本論文では, 反転授業の事前学習のような自習環境におけるプログラミング教育における例題プログラミングを, 1) 文法理解のためのプログラムデバッグ問題, 2) プログラム意味理解のための出力推定問題, の 2 要素に集約し, その例題をテンプレートから自動生成する web アプリケーションを PHP 言語を用いて開発した. さらに, 簡易的な MOOC と組み合わせた実験環境を構築し, その評価により一定の効果を確認した.

1. はじめに

インターネットの普及やスマートフォンの浸透にともない, 様々なコンピュータアプリケーションが身近となり, それらを作成するためのプログラミング教育が様々な段階で重要になってきている. 一般に, Java や C 言語などのプログラミング教育において, 文法や開発ツールを理解するだけでなく, 実際のプログラミングを例題などを使って行い, プログラムを繰り返し作成することが言語習得には必要不可欠であると考えられる.

一方, 新しい教育スタイルのひとつとして MOOC (Massive Open Online Course) を用いた反転授業 (flip teaching) が注目を集めている [1]. 反転授業では, 予習として知識習得型の学習をしておき, 教室においてその知識の定着を図るための演習などを行うものである. プログラミング教育においても反転授業を適用しようという試みはあり, Maher らの文献 [2] においては, web アプリケーション開発の教育をオンラインビデオとクラスでのグループワークを組み合わせ, その評価を行っている.

前述の通り, プログラミング教育においては文法のような知識習得だけでなく, 実際にプログラムを作成する体験がその理解には重要となり, 反転授業に適用する場合は, 予習においても多くの例題プログラミングを行う機会が必要である. しかし, 例題プログラミングを学習者が自分で採点することが難しく, また, 多くの例題を予め作成するのもコストが大きい. 一方, Radošević らは, プログラミング教育におけるオンライン教材の自動生成について研究している [3]. 彼らの手法は部分的には成功しているが, 運用する教員に高いスキルと要求し, 採点の手法に問題がある. また, いくつかの先行研究では課題の自動採点については取り組まれているが, 課題自体の自動生成にはふれられていない [4], [5].

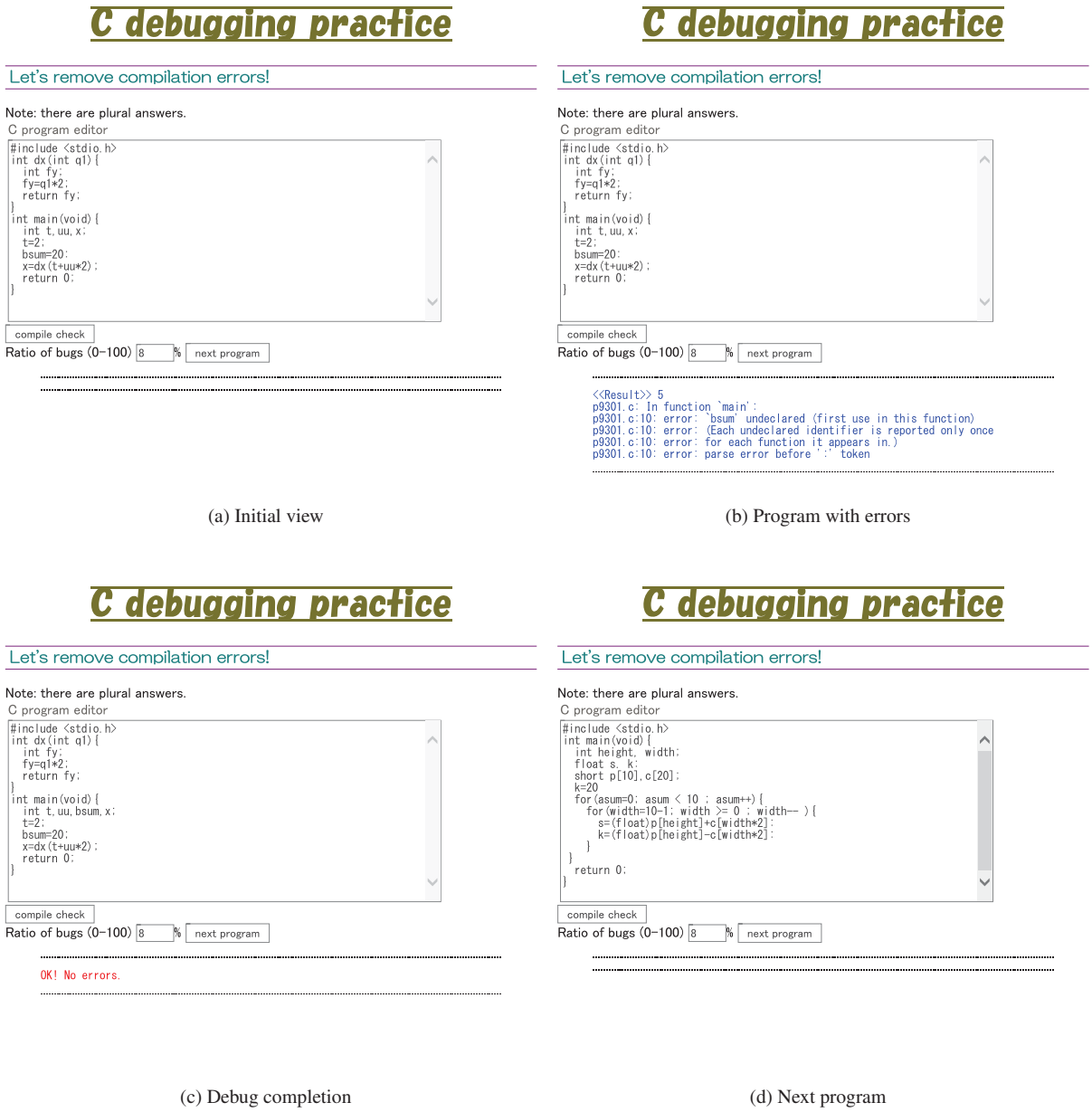
そこで, プログラミング教育における例題プログラミングを, 1) 文法理解のためのプログラムデバッグ問題, 2) プログラム意味理解のための出力推定問題, の 2 要素に集約し, 簡単で採点が容易なレベルの例題のみに注目し, その例題をテンプレートを用いて自動生成する web アプリケーションを, PHP 言語を用いて開発し, 簡易的な MOOC と組み合わせて評価する.

2. プログラミング教育補助教材

プログラミング言語教育の基本は文法の習得である. そこで, 文法理解の補助教材として, バグを含むプログラムを学習者に提示し, それを修正することにより正しいプログラムにする “プログラムデバッグ問題” を用意する. また, プログラム作成においては, 自らが考えるアルゴリズムや解法を正しくプログラムに表現することが必要となる. その能力を涵養するために, “出力推定問題” では, 意味理解の補助教材として, 与えられたプログラムを実行させずにどのような挙動をするかを推定し, 画面に表示される値を答えることを学習者にさせる. なお, 本論文では, C 言語のプログラミング教育での実装例を示すが, 本手法は他のプログラミング言語でも容易に適用可能である. また, プロトタイプシステムは, http://pplinux.is.konan-u.ac.jp/rccE.html においてある.

[1] 甲南大学
Konan University, Okamoto, Kobe 658–8501, Japan
[2] 阪南大学
Hannan University, Amami Higashi, Matsubara, Osaka 580–8502, Japan
[a)] wakatani@konan-u.ac.jp

Vol.2015-CE-132 No.13
Vol.2015-CLE-17 No.13
2015/12/5



(a) Initial view

(b) Program with errors

(c) Debug completion

(d) Next program

図 1　プログラムデバッグ問題
Fig. 1　Syntax practice

2.1　プログラムデバッグ問題

2.1.1　概要

プログラムデバッグ問題の補助教材は, 文法理解の定着を図るためにバグを含むプログラムのデバッグを繰り返し行い, 文法理解の間違いを修正することを目的とする. 図 1 に本補助教材の実行画面を示す.

本補助教材では, まず, 中央の枠内にバグを含むプログラム例が表示される (図 1 (a)). このまま “compile check” ボタンをクリックすると, コンパイルエラーが表示される (図 1 (b)). 学習者は, 理解している文法知識とエラーメッセージを参照してプログラムを修正し, エラーがなくなるまでチェックを続け, 最終的にバグが無くなると文法的に正しいプログラムになる (図 1 (c)). さらに学習を続ける場合は, “next program” ボタンをクリックすると, 新しいバグありプログラムが表示される (図 1 (d)). なお, 新しい問題を提示する際に, バグ率 (ratio of bugs) を変えることにより, バグの含有率を変えることができる. バグ率を 0 にするとバグの無いプログラムが提示され, 100 にすると, 以降で述べるバグのすべては含まれるプログラムとなる. しかし, バグ率が高過ぎるプログラムであれば, 本来の正しいプログラムが見えづらくなるので, デフォルトは 8 に設定している.

2.1.2　教育効果

デバッグにおいては, コンパイラのエラーメッセージを利用する. 文法の理解はエラーメッセージに頼らなくてす

```
$p[$i]= ”#include <stdio.h>
int main(void){
  float __v0;
  int __v1,__v2,__a0[20];
  __v0=__v1+__v2+__a0[2];
  return 0; }”;
```
(a) Program template

```
$vn=array(”i0”,”j0”,”asum”,”bmax”,”cmin”,
”dx”,”ex”,”fy”,”cmax”,”height”,”k”,”bmin”,
”bsum”,”num”,”ok”,”p1”,”q1”,”r0”,”s”,”t”,
”uu”,”vv”,”width”,”x”,”y”,”z”);
$an=array(”zz”,”yy”,”xx”,”wvec”,”vvec”,
”uvec”,”t”,”s”,”r”,”q”,”p”,”o”,”n”,”m”,”l”,
”k”,”j”,”i”,”h”,”g”,”f”,”e”,”d”,”c”,”b”,”a”);
```
(b) Name list

```
$sp[$i][0]=”stdio.h”;
$rp[$i][0][0]=”float”;
$rp[$i][0][1]=”single”;
$rp[$i][1][0]=”;”;
$rp[$i][1][1]=”:”;
```
(c) Bug information

```
#include <stdio.h>
int main(void){
  single y;
  int z,dx,d[20];
  y=k+dx+g[2]:
  return 0;
}
```
(d) Generated program

図 **2**　プログラムデバッグ問題の PHP スクリプト
Fig. 2　PHP script for syntax practice

べきという考え方もあるが, 実践的場面においては, プログラミングはコンパイラを利用することが普通である. したがって, エラーメッセージを参考に, 正しい文法へ変更することにより, 理解し間違えていた文法を正しく理解できるようになる. また, バグ率を変更できるので, より深い理解をしたい場合は, バグ率をあげて教育効果の効率を向上することも可能である. 以上のことより, 教材を読むだけで文法を理解するのではなく, 一定の実践的な例題プログラミングを課題として行うことができ, 文法の定着が図れると期待できる.

2.1.3　実装方法

基本的には PHP 言語を用いた web アプリケーションであるが, 全ての問題を予め用意しておくのではなく, 元となるテンプレートプログラムとそのプログラムをバグを含むようにするための指示のデータを用意し, 学習者から要求に従い, 実行時にバグありプログラムを生成する.

図 2 に PHP スクリプトの一部と生成されたプログラム例を示す. まず, テンプレートとなるプログラムは, 変数名及び配列名を “__” で開始する名前で記載し, 実行時に name list にある名前候補からランダムに当てはめが行われる (図 2 (a), (b)). これにより, 学習者は同じテンプレートであっても同じプログラムであることに気づきにくく, 繰り返し学習に適したプログラムになる. なお, テンプレートは文法理解の進度に合わせて作成し, それぞれの例題のバリエーションのために, 10 から 20 個程度の複数のテンプレートを用意する.

また, バグの生成方法は 3 種類に対応している. まず, 前述の通り, 変数名及び配列名は実行時に当てはめが行われるが, この当てはめをランダムに変更することで, 未宣言変数などのエラーを生成する. 図 2 (a) には 2 種類のパターンが記載されている. “sp” で開始する配列には文字の書き間違えを行う文字列の一覧を記載する. 例えば, stdio.h であれば, 適当に文字を削除したり挿入したりして, stdi.h や stdiio.h のような書き間違えパターンを生成する. 一方, “rp” で開始する配列には文字の書き間違えのパターンを指定しているものである. これは, よくある書き間違えパターンを指定するもので, 例えば, コロンとセミコロンの書き間違えや, float と single の覚え間違え, のようなエラーに対応したものになっている.

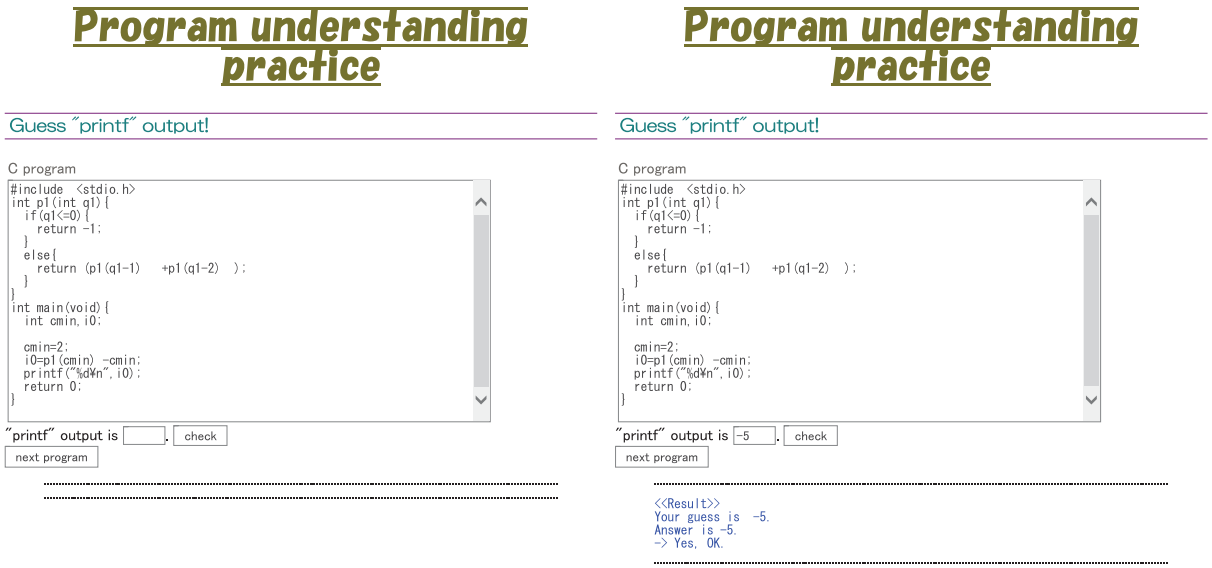
学習者が提示されたバグありプログラムを修正した後, スクリプトは修正後プログラムをユニークなファイル名で保存し, サーバ内のコンパイラでコンパイルし, そのコンパイラからのメッセージを学習者のブラウザに返信する. コンパイラからのメッセージがなければ修正が終了したとして, “OK” の文字列を返す. なお, 使用するコンパイラは C コンパイラであるが, コンパイラを変更し, テンプレートプログラムなどを変更すると, C 言語以外への応用も容易で, 幅広いプログラミング教育に利用可能である.

2.2　出力推定問題

2.2.1　概要

出力推定問題の補助教材は, プログラムの意味理解を図るために結果出力の予測を行い, 正しい意味を理解することを目的とする. 図 3 に本補助教材の実行画面を示す.

本補助教材では, まず, 中央の枠内にプログラムが表示される (図 3 (a)). 学習者はこのプログラムを読み, その意味を理解した上で printf 関数の出力を推定する. 推定した値を回答欄に記入後, “check” ボタンをクリックすると, 実行結果との比較が表示される (図 3 (b)). 表示されるプログラムには, 関数の再帰呼び出しを含むものもあるので暗算での回答が難しいものがあるが, プログラムの意味を理解する能力の涵養を図ることができるものになっている.



(a) Initial view (b) Result guess

図 3 出力推定問題
Fig. 3 Semantic practice

なお, 問題のパラメータ設定によれば, ゼロ割算やメモリアクセスエラーなどを発生させる可能性があるので, 実行時のエラーが生じないように留意する必要がある.

2.2.2 教育効果

繰り返しや条件分岐, 関数呼び出しによるプログラムの動作を把握し, 意味を理解することは正しいプログラム作成に欠かせない要素である. この例題に正しい回答を与えるためには, 学習者はプログラムの動作を考える必要があり, 意味理解を向上することができる. また, 変数名や定数などを随時変更し, 暗算でも計算可能なコンパクトになるような例題を用意することにより, 繰り返し例題を行え, 教育効果も向上すると考えられる.

2.2.3 実装方法

基本的には PHP 言語を用いた web アプリケーションで, プログラムデバッグ問題と同様の作成方法であり, 全ての問題を予め用意しておくのではなく, バリエーションのある問題を生成するために, 元となるテンプレートプログラムと置き換えるパターンを用意し, 学習者から要求に従い, 実行時に問題となるプログラムを生成する.

プログラムデバッグ問題と同様に, 変数名や配列名は name list の中からランダムに選ばれる. 例えば, val1 は 1, 2, 3, 4, 5 と, val2 は 4, 5, 6 と実行時に置き換えられ, 複数のパターンの問題が生成できる. また. 演算子や if 文の条件なども置き換えるようにする. これにより, 学習者は同じテンプレートであっても同じプログラムであることに気づきにくく, 繰り返し学習に適したプログラムになる.

提示されたプログラムの出力を学習者が推定後, スクリプトはプログラムをユニークなファイル名で保存し, サーバ内のコンパイラでコンパイルして実行したのち, 実行結果と学習者の推定値の比較を学習者のブラウザに返信する. なお, 使用するコンパイラは C コンパイラであるが, コンパイラを変更し, テンプレートプログラムなどを変更すると, プログラムデバッグ問題と同様に, C 言語以外への応用も容易で, 本補助教材も幅広いプログラミング教育に利用可能である.

なお, 前述の通り, 文字列の置き換えによりゼロ割算やメモリアクセスエラーなどの実行時エラーが発生する可能性があるので, スクリプトを作成する際にそのような問題を生じさせないようなパラメータ選択にすることに留意する必要がある.

3. 実験と考察

3.1 実験方法

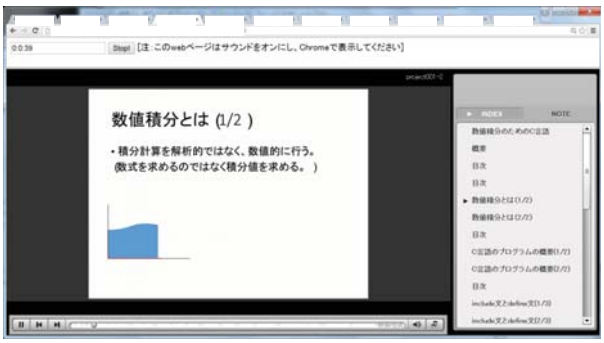


図 4 実験環境
Fig. 4 Evaluation environment

Vol.2015-CE-132 No.13
Vol.2015-CLE-17 No.13
2015/12/5

評価実験では, 問題の対象を決め, 1) 事前学習 (約 12 分), 2) 事前テスト (10 分), 3) 例題による学習 (8 分 (文法理解)+7 分 (プログラム意味理解)), 4) 事後テスト (10 分) とし, 事前テストと事後テストの結果を比較して, 例題による学習の効果を評価するものである.

評価実験で対象とする問題は数値積分の C 言語プログラムで, 事前テストは $f(x) = \frac{1}{x^n}$ の数値積分の C プログラムに後述のいくつかのバグを含んだものとし, 事後テストは事前テストよりも複雑さが若干増すように, $f(x) = x^n$ $(n = 1...10)$ の数値積分と $F(x) = \frac{1}{n+1}x^{n+1}$ $(n = 1...10)$ の不定積分を用いた積分計算を行う C プログラムとしている. 事後テストを若干複雑にしたのは, 事前テストで数値積分のプログラムの理解が進んでいると考えられ, 同程度の難しさであれば事後テストの方が簡単に見える可能性があるからである. 事前テストで用いるプログラムは 40 行で, 事後テストは 50 行である.

それぞれのテストは文法理解 (syntax) とプログラム意味理解 (semantics) に関していくつかのバグを付加しており, その指摘と修正方法を問うものとなっている. syntax のバグについては, 1) stdio.h の綴り間違い, 2) define 文のセミコロン, 3) 関数の仮引数並びにセミコロン付加, 4) 複数の変数宣言のカンマ忘れ, 5) integer キーワード, 6) for 文の条件の中の最後にセミコロン付加, 7) 文末にコロン付加, 8) 文末のセミコロン忘れ, 9) 未定義演算子の使用の間違いを付加している. また, semantics については, 1) 関数の仮引数と関数内の変数名のミスマッチ, 2) printf 文の書式のミスマッチ, 3) for 文の条件のミスマッチ 1(< か <= か), 4) for 文の条件のミスマッチ 2(+1 するか否か) の間違いを付加している.

事前学習のコンテンツは powerpoint を用いて作成する. 内容は数値積分の説明と C 言語プログラミングの概要, 特に事前テスト及び事後テストで問われる内容に関する部分を強調したものとする. powerpoint のノート部分に喋る文言を記述し, STORM Maker を用いて自動的に音声合成した web コンテンツを生成する. また, 事前学習, 事前テスト, 例題学習及び事後テストからなる実験シナリオを Javascript で記述し, Chrome に内蔵の音声合成機能によりガイダンスが音声合成されるようにした web コンテンツを利用する. 図 4 に事前学習を表示している web ページの例を示す.

被験者は, C 言語は一通り学んだ男子学生 (21 才もしくは 22 才)7 名とする. 被験者は既に C 言語は履修しているが, 履修後の C 言語の使用頻度は高くないので, 今回の実験の開始時点では必ずしも文法理解が定着していないという状況である.

3.2 結果と考察

表 1 に評価実験による事前テスト (pre) と事後テスト (post) での正答率の結果を示す. なお, 正答率は, 文法理解 (syntax) のバグのカテゴリー 9 個の中から正しく修正方法を示された割合, プログラム意味理解 (semantics) のバグのカテゴリー 4 個の中から正しく修正方法を示された割合を示す.

表 1　評価結果
Table 1　Evaluation results

	Syntax		Semantics	
	pre	post	pre	post
1	44.4%	55.6%	0.0%	0.0%
2	33.3%	44.4%	0.0%	0.0%
3	33.3%	44.4%	0.0%	25.0%
4	44.4%	55.6%	25.0%	25.0%
5	0.0%	22.2%	50.0%	50.0%
6	44.4%	55.6%	0.0%	0.0%
7	88.9%	88.9%	25.0%	25.0%
total	41.3%	50.8%	14.3%	20.8%

まず, 文法理解 (syntax) 及びプログラム意味理解 (semantics) のいずれも, 全体的には事前テスト (pre) よりも事後テスト (post) の方が成績がアップしており, 例題を用いた学習の効果は確認できる. 正解率は semantics よりも syntax の方が高い. syntax の間違いは文字パターンを見るだけで気づくものが多い. 例えば, "`int main (void;)`" や "`for(i=0;i<DIV_NUM+1;i++;)`" は, 継続的に C 言語プログラミングをしている人はすぐに気づくものである. また, "`integer i;`" のようなものは, 他の言語 (例えば FORTRAN) などにも知識がある場合は, かえって見落とす場合がある. ほぼ全ての被験者において, 事前テストよりも事後テストの方が成績があがっており, プログラムデバッグ問題を用いた文法理解の学習は短時間 (8 分) であっても高まっていると考えられる.

一方, semantics の場合は, 深くプログラムを理解しないとバグは見つけにくい. 例えば, 数値積分をする範囲を `DIV_NUM` 個に分割し, その合計を計算する部分の繰り返しに "`for(i=0;i<DIV_NUM+1;i++;)`" と記載しているが, 実際は "`for(i=0;i<DIV_NUM;i++)`" と修正すべきである. しかし, テスト時間が 10 分であるのでそこまでの理解に至る前に時間切れになった可能性がある. 事後テストで成績が上がった被験者は一人であり, 新たに見つけられたバグは printf 文の書式のバグであった. 全体を通して, for 文の条件のミスマッチのバグを見つけた被験者はいなかった. 深いプログラム理解を進めるための例題プログラムの生成方法には今後さらなる工夫を必要とし, syntax と semantics の学習の時間の配分も今後の検討事項と考えられる.

4. おわりに

本論文では, プログラミング教育における例題プログラミングを, 1) 文法理解のためのプログラムデバッグ問題, 2) プログラム意味理解のための出力推定問題, の 2 要素に集

約し, それを PHP 言語を用いた web アプリケーションとして開発し, その予備評価により一定の効果を確認した.

今後は, プログラミング意味理解の例題のあり方について再考するとともに, 文法理解とプログラミング意味理解の時間配分の検討を行う. また, MOOC との組み合わせによる実用的なシステムとして構築し, 実際のプログラミング教育における実証的な実験をしていき, さらに, C 言語以外の言語へも適用範囲を広げ, プログラミング教育へ貢献していく.

謝辞

本研究の一部は JSPS 科学研究費 (基盤研究 (C) 15K00501, 2015-2017) 及び私立大学等経常費補助金特別補助「大学間連携等による共同研究」による.

参考文献

[1] J. Wesley Baker, “The “Classroom Flip”: using web course management tools to become the guide by the side,” in *Proc. 11th International Conference on College Teaching and Learning*, pp. 9-17, 2000.

[2] M. Maher, H. Lipford and V. Singh, “Flipped classroom strategies using online videos,” http://maryloumaher.net/Pubs/2013pdf/Flipped-Strategies-CEI-Report.pdf, 2013.

[3] D. Radošević, T. Orehovački and Z. Stapić, “Automatic on-line generation of student’s exercises in teaching programming,” in *Proc. Central European Conference on Information and Intelligent Systems*, CD-ROM, 7 pages, 2010.

[4] S. Gupta and S. Dubey, “Automatic assessment of programming assignment,” *Computer Science & Engineering: An International Journal*, vol. 2, no. 1, pp. 315-322, 2012.

[5] R. Queirós, J. Leal, S. Gupta and S. Dubey, “Programming exercises evaluation systems: an interoperability survey,” in *Proc. 4th International Conference on Computer Supported Education*, pp. 83-94, 2012.