プログラムのソースコードを解答とする問題の柔軟な自動採点

阪 春輝*1・四方 雅晴*1・北 英彦*1
Email: 418m234@m.mie-u.ac.jp

*1：三重大学大学院工学研究科電気電子専攻

◎Key Words e ラーニング、プログラム作成、自動採点

1. はじめに

情報化社会の発展に伴って、プログラミング教育の必要性が高まっている。そこで、学生にプログラミング技能を習得させるためには、様々なプログラムを数多く作成させることが重要である。しかし、プログラミングの講義では演習時間が足りないため、作成することができていないという現状がある。講師やTAの数に対して学生の数が多い場合が多く、学生の書いたプログラムひとつひとつに対して講師がコメントを含めたフィードバックを返すとなると講師への負担が大きい。

そこで本研究では、プログラミング演習の授業の方法としてeラーニングシステムを利用することにした。eラーニングシステムは多くの大学が導入しており、その中の有用な機能のひとつに小テストがある。小テストを取り入れることにより演習以外でも学生に多くのプログラムを作成させることができる。しかし、ソースコードのような表現の自由度が高い記述式問題における学生の解答を自動採点することは困難であり、講師による採点への負担が増えるといった課題があった。

著者の所属する研究室の伊藤はソースコードを解答する問題に対して自動採点システムを考案した[1]。問題はソースコードの一部を記述する穴埋め問題とする。採点項目は、講師が採点のときに確認していると思われる構文エラーの有無、動作の正しさ、などである。この自動採点システムを実装し授業で使用した結果、学生に数多くのプログラム作成の問題に取り組ませることができ、その一方、講師の負担は増えないということを確認した。解決すべき課題として、構文エラーのあるものは、動作の正しさを判定できないため 0 点になってしまうことがあげられる。これにより、得点分布が満点側と 0 点側でほぼ二極化していた。人間である講師が手動で採点する場合、軽微な構文エラーに関してはその部分が正しいものとして採点できるのに対して、このシステムでは採点を継続することができないため、初心者に対しては厳しすぎる採点になっている。

本研究は、軽微な構文エラーはシステムが自動的に修正を行い、コンパイル可能で実行できるものに修正して動作の正しさをチェックし点数を与えることができるようにすることを目的とする。これによって、講師が手動で行う場合に近い採点が自動で行えるようになる。

また、このシステムをオープンソースの e ラーニングシステムプラットフォームである Moodle[2]に取り入れることでプログラミングの小テストを行い、その結果のフィードバックを学生と教師へ素早く行うことができるようにし、その効果を測定する。これにより、講師は学生に対して多くのプログラム作成の問題を課すことができ、学生は多くのプログラムを作成することが可能になる。

2. 関連研究

本システムと同じようにプログラミング教育を行う上で自動採点に焦点を当てた研究はいくつか行われている。柳田らは、プログラムとトレース表を用い様々な難度の１行程度の組み合わせの穴埋め問題を作成し、解答後は自動採点を行い学習者に採点結果を示すシステム pgtracer を開発している[3]。pgtracer は学生の解答に対し、プログラムをコンパイル・実行し正解のプログラムの実行結果の比較と、入力された文字列に対応するノードの値との比較という２通りの比較で正誤判定を行うことで自動採点を行っている。しかし、コンパイルがエラーになった場合には結果が表示されないため 0 点となってしまい人による採点結果と比べると採点が厳しい。また、プログラミングのスタイルについて見ていない。

3. 伊藤の自動採点システム

伊藤の作成した自動採点システムについて説明する。三重大学電気電子工学科では Java を用いたプログラミングを学んでいるので、対象は Java で書かれたソースコードとする。ソースコードを解答とする小テストの目的のひとつは学習者に数多くのコードを書かせることにあるので、ソースコード全体を解答させるのではなく、小テストの問題の出題意図であるコアの部分のみを穴埋め問題形式で解答させる。

3.1 採点項目

以下の 4 つの項目に関して自動採点を行う。括弧内は配点の標準の割合であり変更できる。

- 構文エラーの有無（30%）
- 動作の正しさ（60%）
- 課題で指定された機能の使用（上記60%の内10%）
- プログラミングスタイルの適切さ（10%）

3.2 伊藤の自動採点システム運用結果

伊藤の自動採点システムを用いた小テストを2015年度の三重大学工学部電気電子工学科のプログラミング演習の受講者 88 名を対象に実施した。ある課題における得点分布を図 1 に示す。結果は二極化した。原因は、コンパイル不可能な場合は動作の正しさを判定できないためである。例えば、現状の採点では文末につける「;」をある 1 行で付け忘れただけで 90%減点される。

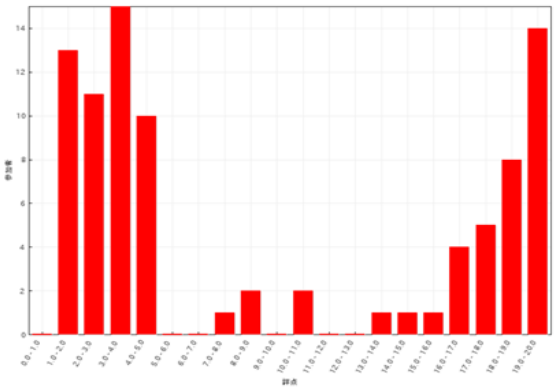

図 1　得点分布

4.　伊藤の自動採点システムの解答の分析

2016 年度のプログラミング演習の受講者 78 名の解答を構文エラーに関して分析した結果を表 1 と表 2 に示す。ソースコードを解答とする小テストの解答を分析した結果、表 1 のように構文エラーを含む解答が約 4 割あり、表 2 に示すように文末の「;」を 1 文字忘れただけのような軽微なものはそのうちの約 3 割であった。構文エラーの重要度は人によって意見が分かれるところであると思うが、本研究では何を書こうとして間違えたのかがその行のみで分かるようなものを軽微なものとして扱うことにした。これは、講師が手動で採点するときに、コードを丁寧に読まなくても判断できる場合に誤りを許容して採点を継続するであろうと考えたからである。

表 1　2016 年度小テスト結果

解答数	構文エラーを含むもの
498	212

表 2　構文エラーの内訳

エラーの種類	件数（%）	修正対象
変数の 2 重宣言	19.8	
文末の「;」忘れ	14.6	○
変数名の綴り間違い	9.9	○
print 文の書式の間違い	7.5	
全角文字の混入	5.2	○
その他	43.0	

5.　自動採点の改良

本研究では、以下のような方法で軽微な構文エラーはシステムがそれらを自動的に修正する。なお、自動的に修正を行った場合は、適当な減点をすることを考えている。修正の結果、解答であるソースコードがコンパイル可能で実行できれば動作の正しさの採点、および、プログラミングスタイルの適切さの採点を行う。

5.1　修正対象

今回修正対象としたものについての対処方法を以下に示す。変数の 2 重宣言は今回数が多かったにも関わらず修正対象としていない理由として、2 重宣言しないように注意しなければならないという意図で作られた問題に対して引っかかった学生が多かったためである。

- 全角文字の混入

  ソースコードの文字列中以外に 1 箇所でも全角文字、特に、全角空白が含まれていると構文エラーになる。小テスト中は、コンパイラを構文チェッカーとして利用できないため全角文字、特に、全角空白を見落とす者がいると思われる。文字列中以外の全角文字は対応する半角文字がある場合には半角文字にシステムが自動的に変換することで、全角文字の混入を除去する。
- 文末の「;」忘れ

  Java などのいくつかのプログラミング言語では、文の最後に「;」をつけることで文の終わりを示す。「;」をコード中でひとつでも忘れれば構文エラーとなり実行できなくなる。コード中のある行が「;」を付けるべき文であるかどうかは文法規則から比較的に簡単に分かるので、文末の「;」を付け忘れていた場合は、システムで自動的に補完し、文末の「;」忘れをなくす。
- 変数名などの綴り間違い

  変数は宣言した段階での綴りが変数名となり、それ以降で使用する際は同じ綴りでなければ構文エラーが発生する。以前の解答を調べた結果では、綴り間違いの種類としては「文字の置き換わり」「文字が多い」「文字の抜け落ち」があった。

  綴り間違いの基準は文字列の類似度を調べる Levenshtein 距離[4]を採用して、1 文字までの間違いを許容することにする。Levenshtein 距離における操作は「挿入」「削除」「置換」「転置」の 4 種類で、これらの操作を 1 度のみ行い修正することができれば綴り間違い、すわなち、軽度の構文エラーとして扱うことにする。

  綴り間違いと判断した場合、システムが自動でそれを正しいものに修正し、綴り間違いによるエラーをなくす。

以上の変更により、軽微な構文エラーで解答全体の点数が 0 点となることを避けることができる。これによって，講師が手動で採点したときの評価に近づけることができると考えられる。

5.2　自動修正の流れ

従来のシステムでは、ソースコードに構文エラーが含まれていた段階で採点を終了していた。

それに対して構文エラーを検出した際に、コンパイラによる構文エラーメッセージを参照し、ソースコードの修正を行う。修正を行う流れとしては図 2 の順番で行う。

この手順で行った理由として、構文エラーの発生する主な原因の優先順位と、優先順位を高い順位修正にすることによって他の構文エラーを発見しやすくするためである。

最初に、ある行に全角文字が含まれていた場合、その行で構文エラーが起こり、他の構文エラーを発見することができなくなるため行う。次に「；(セミコロン)」が抜けている行があると同じように、上手くコンパイルすることができなくなるためである。最後に他 2 つのエラーがなくなった状態で綴り間違いを探し、修正を行う。このような段階を踏むことで、コンパイルエラーを修正しやす

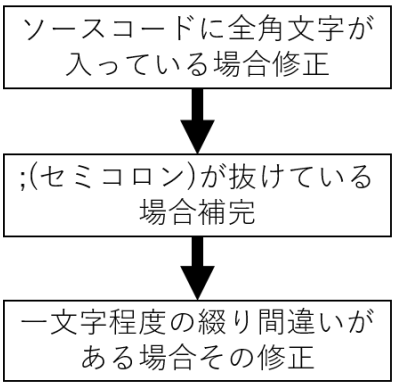

図 2　ソースコードの修正の流れ

くなると考えた。もし、この 3 段階で修正されて構文エラーがなくなれば、ソースコードの動作を評価することができる。この場合の構文点は減点を行う。それと同時にどの修正を行ったかを講師側に表示する。この情報を元に学生への構文エラーのフィードバックについて講師をサポートする。

5.3　システムの構成

今回の採点システムの動きについて構成図を図 3 に載せる。赤線で示したものが今回追加した自動修正システムであり、それ以外は伊藤のシステムを利用する。伊藤のシステムでは、はじめに e ラーニングシステムから学籍番号と解答部分のデータが入った CSV ファイルを入手し、サポートシステムによって Java の雛型ファイルと結合させる。その学生解答部分を含むソースコードをテストケースや配点などの情報を持った XML ファイルを元に自動採点システムを通して採点を行っており、講師の採点における負担を大きく軽減させる効果が得られていた。今回追加した機能としては、自動採点システムで採点を行う前に前節で述べたような構文エラーを含むソースコードに対しては自動修正を行い、その結果を e ラーニングシステムにフィードバックする機能である。また、フィードバックの結果を学生に返す。e ラーニングシステムの外部モジュールとして作成し、導入することで手動採点の必要がなく、結果が返ってくる。実際の Moodle 上における自動採点モジュールの画面を図 4 に示す。

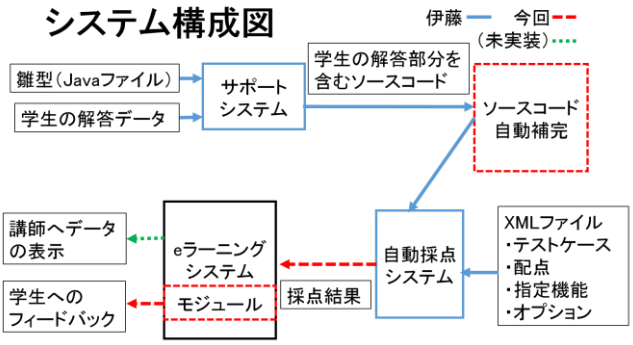

図 3　今回の自動採点システム構成図

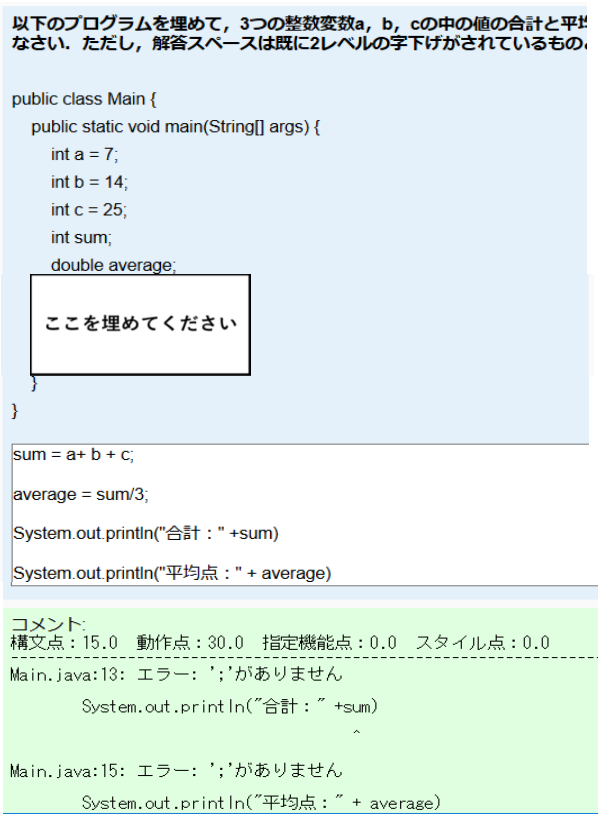

図 4　フィードバックされた様子

6.　採点結果の検討

2016 年度のプログラミング演習の受講者 78 名の解答データを使用し、今回のシステムで自動採点を再び行った。その結果を表 3 に示す。

表 3　動作結果

	指摘数	期待通りの動作	動作失敗
#1	12	12	0
#2	12	7	5
#3	18	11	7
#4	11	4	7
#5	15	9	6
#6	11	9	2
#7	9	4	5
#8	6	2	4
#9	7	6	1
#10	3	2	1
#11	3	3	0
#12	0	0	0
割合(%)		64.5	36.4

6.1　採点結果の分類

期待通りの動作とは、修正対象のソースコードに対して修正・補完が行われ、また修正すべきソースコードであったものを正確に修正できていたものである。

修正に失敗したものはエラーメッセージを元に修正を試みようとしたが、メッセージがエラー以外の箇所を指摘していたため、修正する必要がないものを修正し、修正

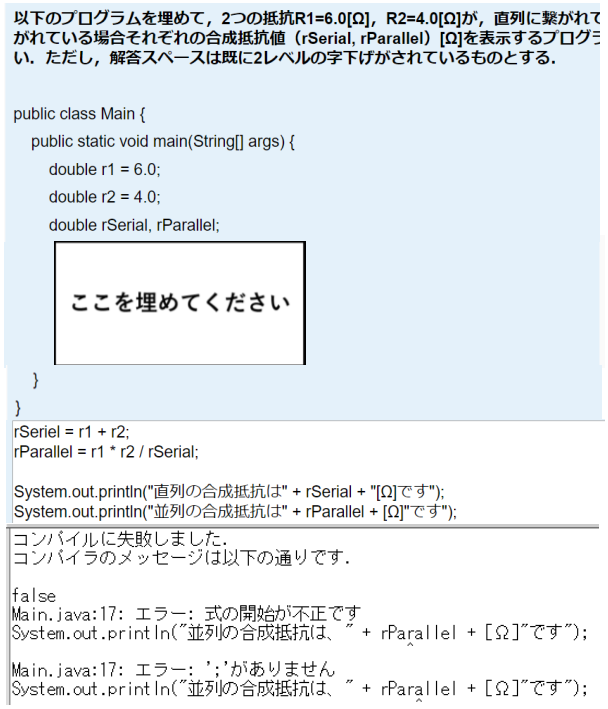

図 5　修正失敗の例

すべきところを修正できなかった数である。その一例を図 5 に示す。構文エラーの原因としては print 文中の文字列の「”（ダブルクォーテーション）」開き忘れである。しかしコンパイラは変数名と文字列をひとまとまりと認識したため、print 文の行に「；（セミコロン）」を忘れていると指摘している。そのためセミコロンを補完するようにシステムが動いたため正しく修正されておらず失敗したとカウントした。

6.2　採点結果の考察

主に修正に失敗した原因として、構文エラーメッセージを元に修正を行っているため条件としている文字列を含むメッセージがある場合誤った動作をしてしまう。一番多かった修正間違いとして挙げられるのがセミコロンの補完だが、その多くは変数名の間にスペースがあるためや、文字列の範囲を示す「“（ダブルクォーテーション）」の抜けなどによりコンパイラが文字列の判定ができなくなるなどといったプログラムとしての繋がりが途切れた段階でひとまとまりのコードと判別してしまうためセミコロンが抜けていると判断していると考えられる。他にも、print 文の「“”（ダブルクォーテーション）」を補完することは技術的に難しく、今回は修正の対象外であったが、連鎖的に他のエラーだと誤認識してしまうため、対策が必要である。今回は修正対象としなかったが、System 文の綴り間違いは変数間違いと同様にレーベンシュタイン距離[3]が 1 文字以内のものについては修正対象にすることとする。

7.　今後の課題

今回の研究では軽微ではない構文エラーとして採点を行ったが、解答を分析した中でもいくつか修正すべき内容か検討を必要とするコードの種類を下記に示す。どれもプログラミングを理解した上で、単純な間違いではないかと考えられるため修正することを検討する。

- 1 か所のみの print 関数の文字列の連結をする「+（プラス）」記号が抜け
- 入力プロンプトを表示する際の「“（ダブルクォーテーション）」での囲い忘れ
- for 文の条件式の区切りが「；（セミコロン）」ではなく「：（コロン）」や「.（ピリオド）」

反対に、今回の修正条件のため修正されたが、それが正しかったか判断に迷うものが見受けられた。それは全ての行に渡る「；（セミコロン）」の抜け落ちである。こちらは、単なるケアレスミスなのか知らないためによるものなのかの判断が難しい。このような場合に対しては今後検討する。

8.　まとめ

学生がプログラミング技能を向上させるための演習時間が足りないという現状がある。そこで、多くの大学でも導入されている e ラーニングシステムの小テスト機能をプログラミング科目に導入することで時間不足の問題を解決することを考えた。しかし、ソースコードのように解答の表現の自由度の高い記述式問題の解答を自動採点することは困難であり採点における講師の負担は非常に大きなものとなってしまう。

伊藤はソースコードの自動採点システムを提案開発し、導入したことで、講師が採点の際に動作テスト以外の観点で自動採点することができた。また、採点における講師の負担が大きく軽減されることが期待できる。しかし、構文エラーを含む問題に対しては 0 点になるといったプログラミング初学者に対して厳しい採点であった。

そこで今回は学生の解答を分析した結果、提出できている解答数のうち約 42%も構文エラーを含んでいた。どのような種類があるか調査した中で、講師が評価した場合、部分点をもらえるような軽微なエラーを 3 つに絞った。今回修正対象となるエラーを含んでいたものは構文エラーを含む件数のうち約 34%あった。これらに関しては 1 件以外修正することができ、プログラムの動作まで評価することができた。

今回提案する自動修正システムを採用することで軽微な構文エラーを含む学生のソースコードを採点することができ、プログラムの動作を評価することができた。また、e ラーニングシステムに取り込み、学生に素早いフィードバックを返すことで学生は様々なプログラムを作成することができるようになる。また、講師に対しては学生の解答状況を把握するページを表示し、講師は必要があれば次の授業の際に注意喚起することができると考える。

参考文献

(1)　伊藤隼人、北英彦：e ラーニングシステムの小テストにおけるソースコードを解答とする問題の自動採点、コンピュータ利用教育協議会、CIEC 春季研究会 2016（2016）
(2)　「Moodle」https://moodle.org/（2018 年 1 月参照）
(3)　柳田崚、太田康介、大月美佳、掛下哲郎：穴埋め問題を用いたプログラミング教育支援ツール pgtracer の運用実験、情報処理学会、情報処理学会論文誌教育とコンピュータ（TCE） 2(2), 20-36, 2016
(4)　Guy Rutenberg: https://www.guyrutenberg.com/2008/12/15/damerau-levenshtein-distance-in-python/（参照 2017 年 5 月）