修士論文

# e ラーニングシステムにおける ソースコードを解答とする問題の 自動採点に関する研究

平成 27 年度修了

三重大学大学院工学研究科

博士前期課程　電気電子工学専攻

伊藤　隼人

# 目次

# 1章　はじめに

近年の情報化社会の発展に伴い，プログラミング教育の必要性が高まっており，工学系の大学や学部のみならず，文系の大学においてもプログラミングは必修科目として教育されている[1-2]．プログラミング演習の授業では，講師は学習者に対してプログラムを作成する課題を与えて，学習者はプログラムの作成を行う．学生がプログラミング技能を向上させるためには，様々なプログラムを数多く作成することが重要である．しかし，プログラミングの講義では，演習時間が足りず，それができていないという現状がある．

この問題点の解決手段として，e ラーニングシステムを利用することが考えられる．e ラーニングシステムは，近年多くの大学で動画を含む教材の提供，学生と講師や学生間でのコミュニケーション，自己学習機会の提供などを目的として導入されている．その中の有用なもののひとつに小テスト機能がある．この小テスト機能を用いて，演習だけでなく小テストを取り入れることにより，学生が多くのプログラム作成問題を解くことができると考えられる．実際に，プログラミング科目にも e ラーニングシステムが導入されており，小テスト機能が利用されている．

しかし，従来のオンラインテストの標準モジュールでは，一般に，多肢選択か文字列として完全に一致するものしか自動採点することができず，ソースコードのような解答の表現の自由度の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った．したがって，プログラミング演習科目において，講師は学生の解答を一つずつ見て，手動で採点することしかできなかった．すなわち，e ラーニングシステムにおける，学生の解答となるソースコードの扱われ方は，紙に解答を記述して提出する場合とそれほど大きな違いはなく，採点における講師の負担が非常に大きなものとなってしまう．

その問題を解決する先行研究として，後に紹介する Online Judge[3]や東京電機大学のシステム[4]などが研究・開発されている．どちらも自動で動作テストを行うシステムであるが，Online Judge における動作テストの基準を満たすことは，初心者にとって難しく，東京電機大学のシステムにおける正規表現を用いたテストケースの設定は講師にとって煩わしい．また，どちらのシステムも基本的には動作しか見ていない．

本研究では，プログラミング科目におけるソースコードを解答とする問題

を対象として，学生の解答となるソースコードを自動採点する方法を検討し，それを実現するシステムを考案する．具体的には，講師が採点するであろう構文エラーの有無，動作の正しさ，指定機能の使用・未使用，プログラミングスタイルの適切さのチェックを行う．動作の正しさのチェックに関しては，同研究室の望月が2004年に提案した，初心者向きに出力形式を緩めた手法[5]を改良して用いる．

本論文の構成は以下のとおりである．2 章でプログラミング演習の現状を交えた研究背景について，3 章で先行研究について述べる．4 章でプログラムを採点する際に見る採点基準について，5 章で自動採点が可能な項目について述べる．6 章で動作テストの手法について述べる．7 章で本研究で提案する自動採点システムについて，8 章でシステムの有用性検証実験の結果・考察を述べる．9 章で本論文をまとめる．

# 2章　研究背景

この章では，教育現場におけるプログラミング演習科目における現状を示し，本研究の背景を述べる．

## 2.1　プログラミング演習科目における現状

プログラミング演習科目において，e ラーニングシステムを導入することによって，学習の効率を上げるという効果が期待できる．実際に，三重大学工学部電気電子工学科のプログラミング演習科目では，e ラーニングシステムの 1 つである Moodle[6]が利用されている．しかし，プログラミング演習科目における e ラーニングシステムの使われ方は，演習としてプログラム作成問題を出して学生にプログラムを書かせ，図 1 に示すように，その解答をファイル形式で提出してもらい，それを講師が一つ一つ手動で採点するというものである．これでは，学生はインターネットが繋がる環境であればいつでも提出できる，解答であるソースコードを講師がコンパイラにかけやすいなどといった利点はあるものの，紙にソースコードを記述して提出する場合とそれほど大きな違いはない．

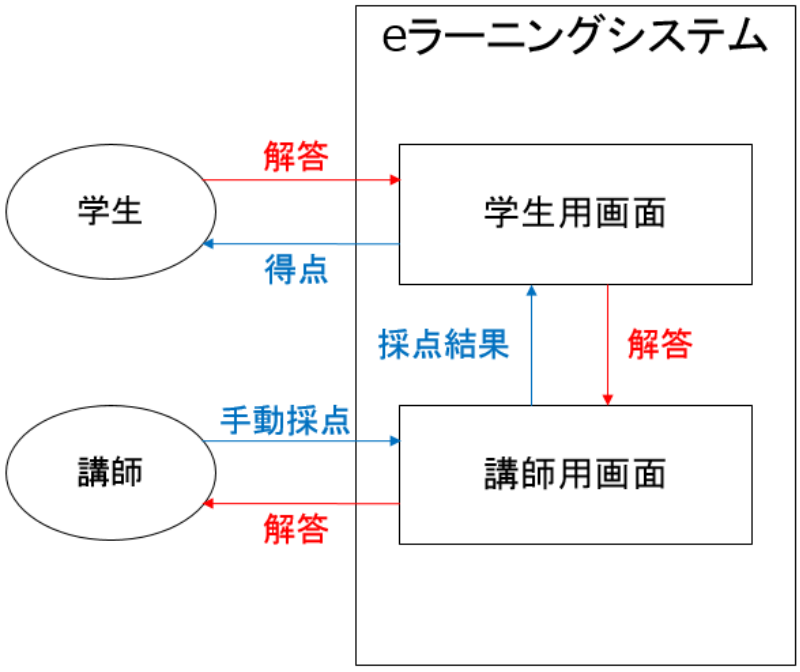


図 1　ソースコードの採点（現状）

また，このような演習形式では，学生がプログラム作成問題を数多くこなすことができないという欠点がある．実際にプログラミング演習Ⅰ・Ⅱの中で演習課題をいくつか解いてもらっているが，表 1 に示すように，1 コマ（90 分）の講義の中で学生が提出できた課題数は，平均 1.4 問と非常に少なく，学生によっては 1 問も解けていない．また，表 2 に示すように，その日の課題 4 つに対して解答を提出した人数を見ても，最後の課題まで辿り着くことができた学生が非常に少ないことがわかる．これでは，学生のプログラミング技能が向上しづらいと考えられる．プログラミング技能の向上には，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が，講師にとって非常に手間がかかる．

表 1　講義（90 分）中に解答が提出された演習問題数

| 用意した問題数 | 最大 | 最小 | 平均 |
|---|---|---|---|
| 4 問 | 4 問 | 0 問 | 1.4 問 |

表 2　各演習課題に対して解答を提出した学生数（全 79 名）

| 演習課題 | 解答提出人数（名） |
|---|---|
| 1 問目 | 78 |
| 2 問目 | 24 |
| 3 問目 | 9 |
| 4 問目 | 3 |

### 2.2　小テスト機能の活用

三重大学で導入されている Moodle をはじめ, Blackboard Learn[7], Maple T.A.[8]などの e ラーニングシステム（または，LMS: Learning Management System）と呼ばれるものには，小テスト機能が存在する．その中で多肢選択問題，短答記述式問題，作文問題（エッセイ問題）など，様々な形式の問題を作成することができる．この小テスト機能を活用すれば，学生にプログラム作成問題を数多くこなしてもらうことができる．プログラム作成問題においては，学生にソースコード全体，またはソースコードのコアとなる部分のみを解答させるために，作文形式が有用である．

しかし，表現の自由度が高い作文形式の問題における解答は，多肢選択形

式，短答記述形式の問題における解答とは違って，自動で採点することができない．これでは，図 1 と変わらず，なおかつ採点しなければならない学生の解答であるソースコードの数も増加してしまい，回収を除いては，講師の負担が非常に大きくなってしまう．

### 2.3　数式解答評価システム STACK の利用

STACK（System for Teaching and Assessment using Computer algebra Kernel）[9]とは，Moodle と連携した数式解答評価システムであり，学生の数式による解答を受け付け，正誤評価・自動採点をするシステムである．ソースコードと数式には，形が一致している部分や似ている部分があることから，STACK をプログラミング演習に利用することについて検討した[10]．しかし，数式と同じ形である条件式や代入文などしか扱うことができず，図 2 のように，非常に限られた範囲で，短答記述形式の問題しか作成することができなかった．また，条件式や代入文に関しても，自然な形では扱えない場合がある．例えば，「tmp = a」という代入文は， STACK の内部では，代入文ではなく等式として扱われる．そのため，講師が慣れるまでに時間がかかるという問題があると考えた．

次のソースコードの穴を埋め、配列arrayに1~10の数値を格納し、表示するプログラムを完成させよ。

```
#include <stdio.h>
#define MAX 10

int main(void)
{
    int i = 0;
    int array[MAX];

    while(                    ) {
        array[i] = i + 1;
        printf("array[%d] = %d\n", i, array[i]);
        i++;
    }

    return (0);
}
```

図 2　STACK で作成できる問題の例（C 言語）

# 3章　先行研究

本章では，ソースコードの自動採点に関する先行研究として，本研究で提案するシステムと運用方法が近い 2 つを紹介する．

## 3.1　Online Judge[3]

Moodle のプラグインモジュールであり，図 3 に示すように，学生が提出したソースコードを，設定したテストケースを用いて，一般的なソフトウェア開発で用いられる厳密な動作テストを行うシステムである．図 4 に示すように，出力が 1 文字単位で厳密に一致しなければ正解とならない．これは，プログラミング初心者には難しい．例えば，初心者の場合，図 5 に示すように，入力プロンプトの「：」の前後に余分なスペースを入れてしまったり，句読点やコンマ，ピリオドを統一していなかったりする．対象言語は，C，C++，Java など 40 以上の言語である．

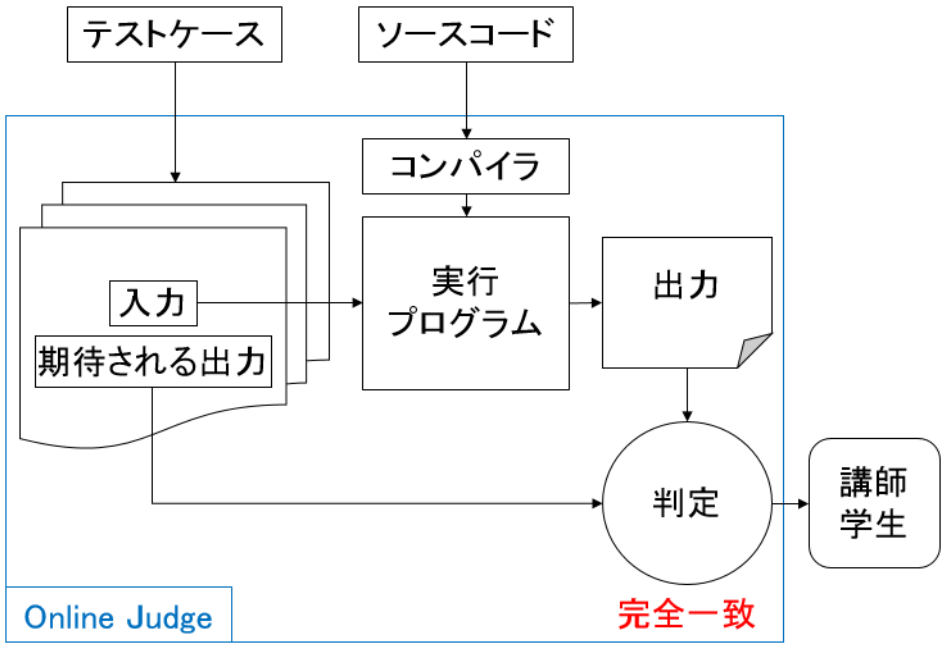


図 3　Online Judge

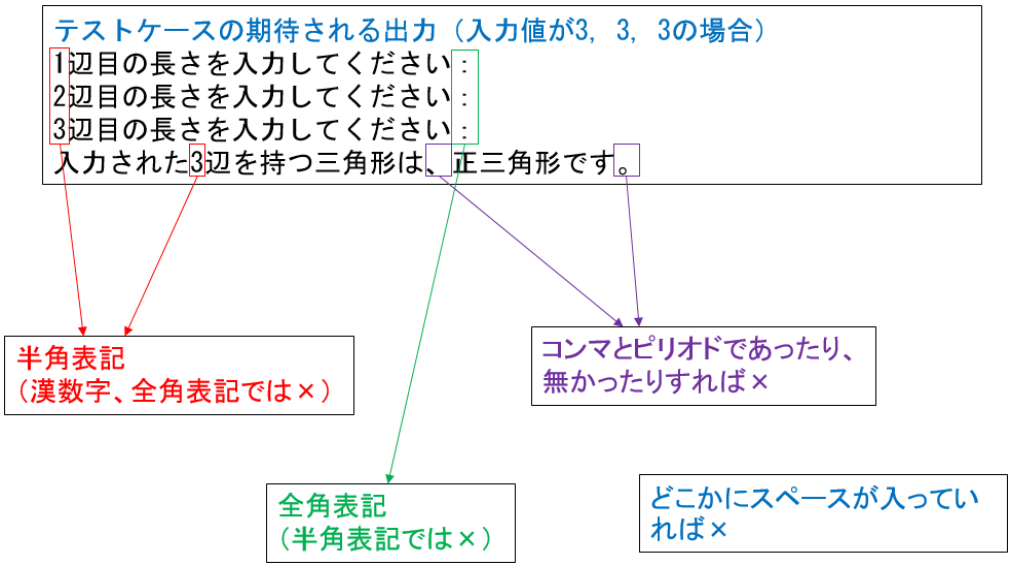


図 4　厳密なソフトウェアテスト

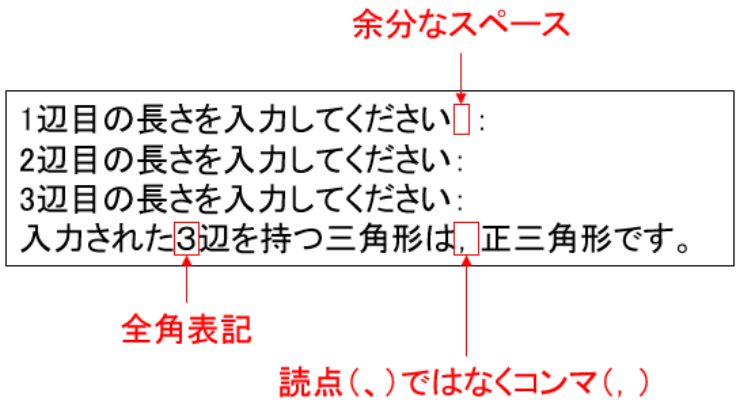


図 5　初心者がやりがちな出力における小さなミス

### 3.2　東京電機大学のシステム[4]

図 6 にシステム概要図を示す．Online Judge 同様，プログラムの自動テストをする．また，類似度算出による盗用防止機能もある．出力の書式のチェックを正規表現を用いて緩めることができる．しかし，出力の全パターンを正規表現で表すのは煩わしい．その例として，図 7 に，入力された三辺の長さからその三角形の種類を判定するプログラムにおける正規表現を使用したテストケースの設定の一部を示す．図 7 からわかるように，一部でもこれだけ書かなくてはならず，手間がかかる．対象言語は Java である．

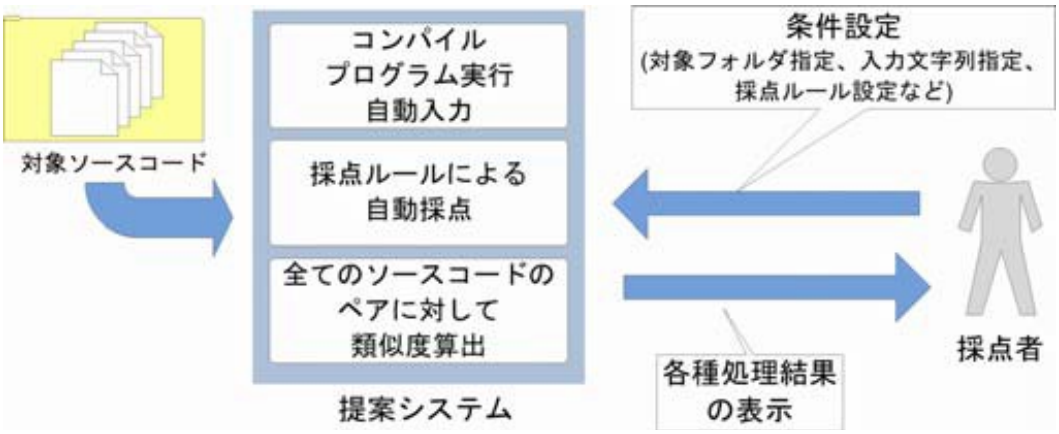


図 6　東京電機大学のシステム

テストケース1
入力：3，3，3（正三角形）
出力：
[11一]辺目の長さを入力して(くだ|下)さい[。．．]?[：：]?[¥n]?
[22二]辺目の長さを入力して(くだ|下)さい[。．．]?[：：]?[¥n]?
[33三]辺目の長さを入力して(くだ|下)さい[。．．]?[：：]?[¥n]?
(入力された[33三]辺を持つ三角形は)?正三角形(です)?[。．．]?[¥n]?

テストケース2
入力：3，4，5（直角三角形）
出力：
[11一]辺目の長さを入力して(くだ|下)さい[。．．]?[：：]?[¥n]?
[22二]辺目の長さを入力して(くだ|下)さい[。．．]?[：：]?[¥n]?
[33三]辺目の長さを入力して(くだ|下)さい[。．．]?[：：]?[¥n]?
(入力された[33三]辺を持つ三角形は)?直角三角形(です)?[。．．]?[¥n]?

：

図 7　正規表現によるテストケース設定

### 3.3　望月の動作テスト手法[5]

望月は，プログラミングの初心者にも課題の目的を達成するプログラムを作成できるように，出力の書式の指定を緩めたテストを提案している．具体的には，図 8 で示すように，出力全体ではなく，出力に必ず含まれるべき文字列（以降，必須文字列と呼ぶ）のみを指定している．プログラムの出力にこの必須文字列が含まれていれば動作成功とみなされる．本研究では，この望月の動作テスト手法を基にした手法を用いる．

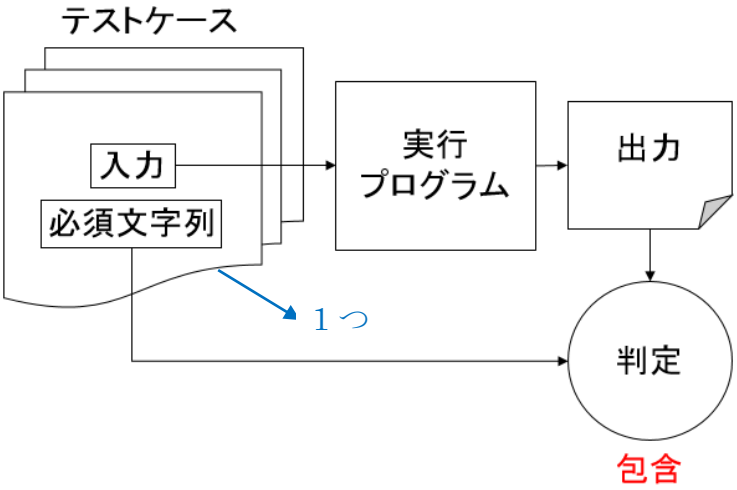


図 8　望月の動作テスト手法

一般には，テストケースとは，テストのための入力と，プログラムが正しく動作した時の出力の組のことである．しかし，ここでは，テストのための入力と，必須文字列の組をテストケースと定義する．また，問題の目的以外の細かい点は，間違いの検出の対象としないこととする．間違い検出の対象としない項目は，以下のものとする．

- 全角・半角の違い
- 空白の有無
- 句読点（またはコンマ，ピリオド）の有無
- 改行の有無

この手法を用いた動作テストにおいて，動作成功と判定される場合と失敗と判定される場合の例を図 9 に示す．

問題点は，テストケース 1 つにつき，必須文字列を 1 つだけしか設定できないことである．例えば，図 9 において，必須文字列「50」のみを設定している場合，単位は「cm」でなくても動作成功とみなされてしまう．だからといって必須文字列を「50cm」にすると図 9 の動作成功例の 3 番目のように，単位に括弧をつけた出力がされた場合，間に「[」が入っており，必須文字列「50cm」が含まれていないため，動作失敗とみなされてしまう．

また，空白の有無を完全に間違い検出の対象としていないことも問題点として挙げられる．出力に空白が必須となるプログラムを作成する問題も存在する．例えば，図 10 で示すような問題の場合，空白（スペース）も出力におけるキーワードとなるため，間違い検出の対象とするべきである．

期待される出力（必須文字列）

50

動作成功例

50 cm

５０ cm

50 [cm]です.

５０ ｃｍ

動作失敗例

五十センチメートル

漢字表記なので×

40 cm

数値が違うので×

図 9　望月の手法での判定の例

0～9の数字を横1列に表示するプログラムを作成せよ.
ただし，各数字の間にスペースを1つ入れること.

図 10　出力に空白が必須となるプログラム作成問題

## 4章　採点基準

本研究を進めるにあたって，プログラミング演習科目を受け持つ講師が，学生の解答を採点する際に，どのような基準を見て採点するのかを検討する必要がある．なお，採点対象としては，プログラミング初心者が作成したプログラムを想定する．実際にプログラミング演習科目を受け持つ講師と話し合い，検討を重ねた結果，表 3 で示すような基準を見れば，適切な採点ができると考えた．

表 3　プログラムの採点基準

| 採点基準 | | 説明 |
|---|---|---|
| 構文エラーの有無 | | コンパイラにかけてもエラーが出ないか |
| 問題の目的通りの動作をするか | | 期待される出力がされるか |
| 問題の意図通りのコードが書けているか | | 問題中に指定された機能（for 文，配列など）が使用されているか |
| プログラミングスタイル（可読性） | 字下げ･空白 | ● 適切なレベル・量で字下げがされているか<br>● 適切な位置に空白が入っているか |
| | 名前の書き方・付け方 | ● 変数名・メソッド名などが適切な書き方（キャメルケース，スネークケースなど）がされているか<br>例：キャメルケース sampleName<br>スネークケース sample_name<br>● 何を表す変数･メソッドなのかがわかる単語が使われているか |
| | コメントの書き方 | ● コメントを付けている場所が適切かどうか<br>● コメントの内容が適切かどうか |
| 効率 | | 効率のいいプログラムになっているか（探索やソートなど大量のデータを扱うプログラムの場合の計算方法など） |
| その他 | | プログラムの動作に不要なコード（使用されていない変数の宣言など）が入っていないか |

# 5章　自動採点

本研究では，採点における講師の負担を軽減することを目的として，ソースコードの採点をどこまで自動化できるかについて検討する．また，実際にそれらの自動採点を行うシステムを考案する．対象となる言語は Java とし，問題はプログラム全体を解答させる問題，図 11 に示すような，ソースコードのコアとなる部分のみを解答させる問題を想定する．

4 章で紹介した採点基準の中で，自動で採点できそうな項目を検討して，以下の 4 つの項目に着目し，自動採点する方法を考えた．効率に関しては自動採点が困難であるため，今後の課題とする．

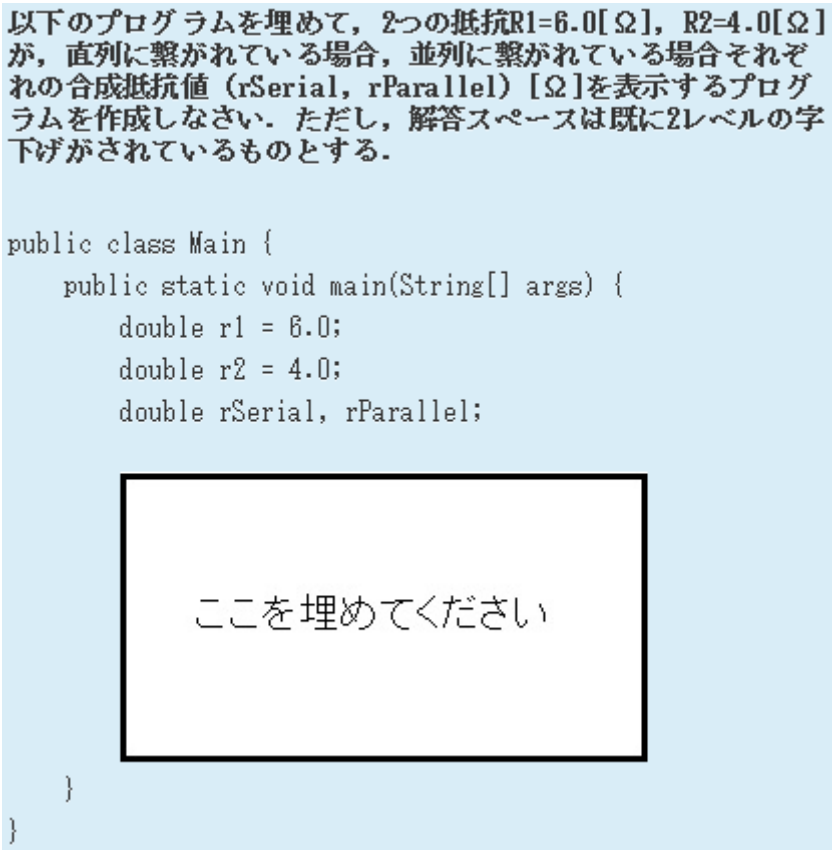


図 11　コアとなる部分のみを解答させる問題

## 5.1　構文エラーの有無

構文エラーの有無は，コンパイラを呼び出して学生の解答ソースコードをコンパイルすることによってチェックする．

### 5.2　動作の正しさ

各問題にいくつかのテストケースを講師が用意し，そのテストケースを満たすかどうかをテストするブラックボックステストを用いる．

ただし，ソフトウェア開発で用いるテストではなく，6 章で説明するプログラミング初心者向きに出力に関する条件を緩めたテストを用いる．

### 5.3　指定機能の使用・未使用

問題文中に「if 文を使って～」や「キーボードから読み込み～」などといった指定がある場合に，それに従っているかどうかをチェックする．それぞれの機能としてではなく，出力文字列や変数名の一部として含まれていても使用しているとは認められない．ここで使用・未使用判定ができる項目を以下に示す．

～制御文～

- if 文
- for 文
- while 文
- switch 文

～その他～

- 配列
- Scanner（キーボードから入力値を読み込む問題の場合に，変数に直接値を代入していないかを判定するため）

### 5.4　プログラミングスタイルの適切さ

プログラミングスタイルが適切かどうかを判定する．適切なプログラミングスタイルの基準として，永和システムマネジメントの「Java コーディング規約」[11]を用いる．判定方法としては，同研究室の上村の手法[12]に習い，Artistic Style[13]というコード整形ツールを用いる．Artistic Style とは，GNU Lesser General Public ライセンスで書かれたプログラムで C, C++, C++ / CLI, Objective-C, C#, Java といったプログラミング言語のソースコードのインデントや整形を行えるツールである．このツールはコマンドライン上で実行でき，自身で引数を与えることで様々なスタイルに整形することができる．図 12 に例を示す．

Artistic Style はあくまでコード整形ツールであり，スタイルが適切かどうかを判定するものではない．そのため，ここでは整形前のコードと整形後のコードを比較し，一致するか否かを判定することによって，適切かどうかを判定する．一箇所でも間違いがあれば点数を与えない，○×形式の判定とする．

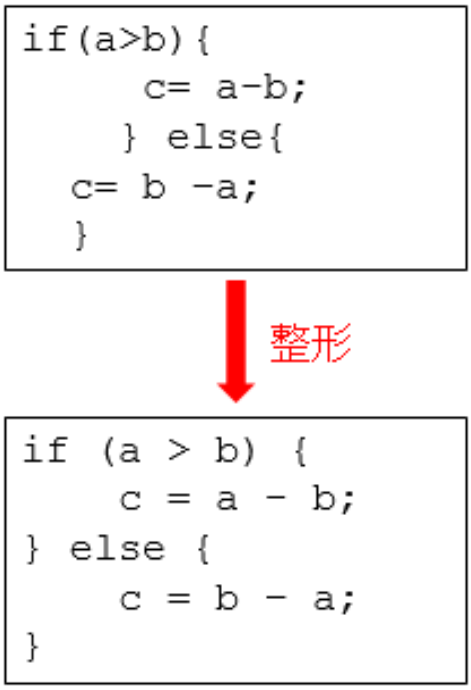


図 12　Artistic Style によるコード整形の例（一部）

# 6章　動作テスト方法

テストを自動化するにあたって，プログラミング初心者の場合には，テストケースの出力値と一字一句一致するプログラムを作成することは難しいという問題点がある．例えば，図 13 のような，標準体重表示プログラムを作成させる問題では，出力の「kg」を全角文字にしたり，「：」記号の前や後に余計な空白や改行を入れたりする．従来の自動テストでは，このような細かい違いによってエラーと判定されてしまう．この問題点を解決する手段としては，問題の中で出力の書式を厳密に指定するか，出力の書式の条件を緩めるかという方法が考えられる．しかし，実際のプログラミングの教科書・参考書の問題を見ると，図 14 に示すように，出力の書式など一切指定されていない問題が多い．

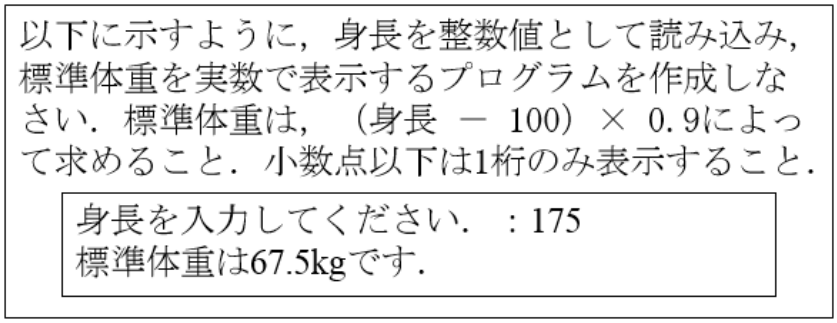


図 13　標準体重表示プログラム作成問題

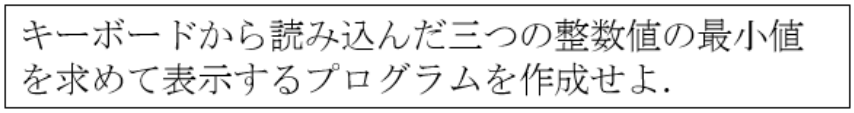


図 14　教科書のプログラム作成問題[14]

## 6.1　厳密な動作テスト手法

問題の中で出力の書式を厳密に指定すると，こと細かに指定する必要があり，非常に煩わしい．また，初心者には難しく，時間もかかると考えられる．例として，図 14 の問題を，出力の書式を厳密に指定した形にすると，図 15 のようになる．実際の教科書では，図 14 のような問題は多くはない．したがって，本研究では出力の書式の条件を緩める手法を用いて動作テストを行う．

キーボードから読み込んだ三つの整数値の最小値を求めて表示するプログラムを作成せよ．空白は入れず「：」は全角とすること．出力書式以外の文字列は出力しないこと．行の最後に改行を出力すること．

【出力書式】

最大値：12

図 15　出力の書式を厳密に指定した問題の例

この手法を用いた動作テストにおいて，動作成功と判定される場合と失敗と判定される場合の例を図 16 に示す．

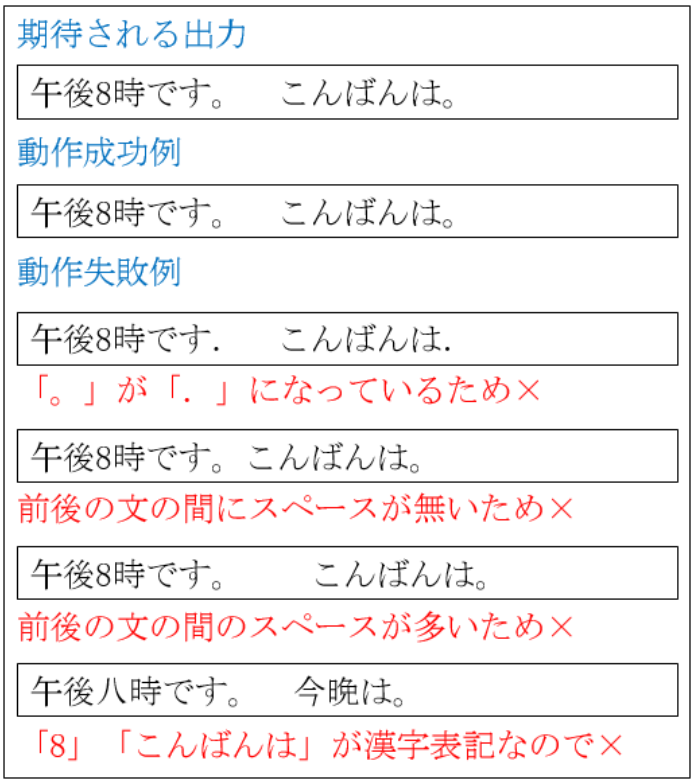


図 16　厳密な手法での判定の例

### 6.2　本研究における望月の手法からの変更点

3.3 節で述べたように，本研究では望月の動作テスト手法を基にした，必須文字列のみを判定に用いた動作テストを取り入れる．

望月の手法を本研究で用いるに当たり，より適切な動作テストができるように，その手法にいくつかの変更を施した．以下にその変更点を示す．

① 必須文字列の複数設定

望月の手法では，テストケース 1 つにつき，必須文字列を 1 つだけしか設定できなかった．本研究では，テストケース 1 つにつき，複数の必須文字列を設定できるように変更した．

例えば，図 13 の問題において，必須文字列として「67.5」と「kg」を設定することにより，単位に括弧をつけた「67.5[kg]」と出力された時も正解とすることができる．

② 出力順の固定

必須文字列を複数設定できるようにしたことにより，不自然な順番で出力されているにも関わらず正解となってしまう可能性がある．そこで，設定した必須文字列の出力順を固定できるようにして，出力に必須文字列が全て含まれていても，順番が違えば不正解とすることができるようにした．

例えば，図 13 の問題において，必須文字列「67.5」と「kg」の順番を固定すれば，「kg67.5」などと出力されても不正解となる．

③ 空白の有無検出のオプション化

望月の手法では，完全に空白の有無は間違い検出の対象としていなかったが，本研究では，オプションで設定することにより，空白の有無も間違い検出の対象とできるようにした．

# 7章　ソースコードの自動採点システム

実際に 5 章で述べたような項目について自動採点を行うシステムを提案する．フィルターのような形で，e ラーニングシステムの外部モジュールとして作成し，導入することにより，図 17 に示すように，図 1 に示した従来のものと比べて，手動採点の必要が無く，結果が返ってくるだけとなり，講師の採点における負担を大きく軽減させることが期待できる．

また，問題が図 11 に示すような形式である場合があるため，雛形の部分と学生の解答部分を結合するサポートシステムについても提案する．

この自動採点システムが受け取るものと渡すものを以下と図 18 に示す．

- 受け取るもの
  - 学生の解答部分を含むソースコード
  - XML ファイル
    - テストケース
    - 指定機能
    - 配点
    - オプション（コード最低行数，空白削除機能，出力順固定機能）
- 渡すもの
  - 学生の解答部分を含むソースコード
  - 採点結果
  - XML ファイル
    - テストケース
    - 指定機能
    - 配点
    - オプション（コード最低行数，空白削除機能，出力順固定機能）

また，この自動採点システムにおいて，学生の解答をどのように評価・採点しているかを表す流れを図 19 に示す．

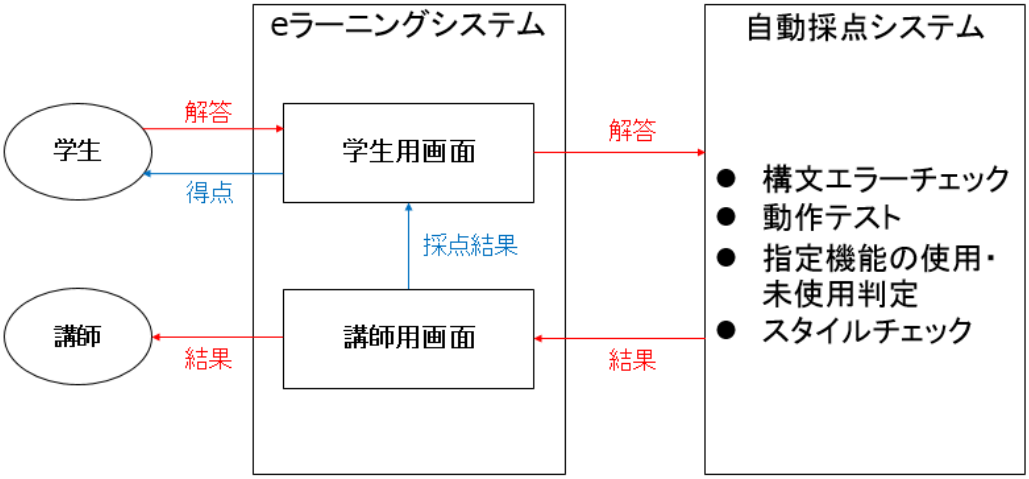


図 17　自動採点システムを使った採点

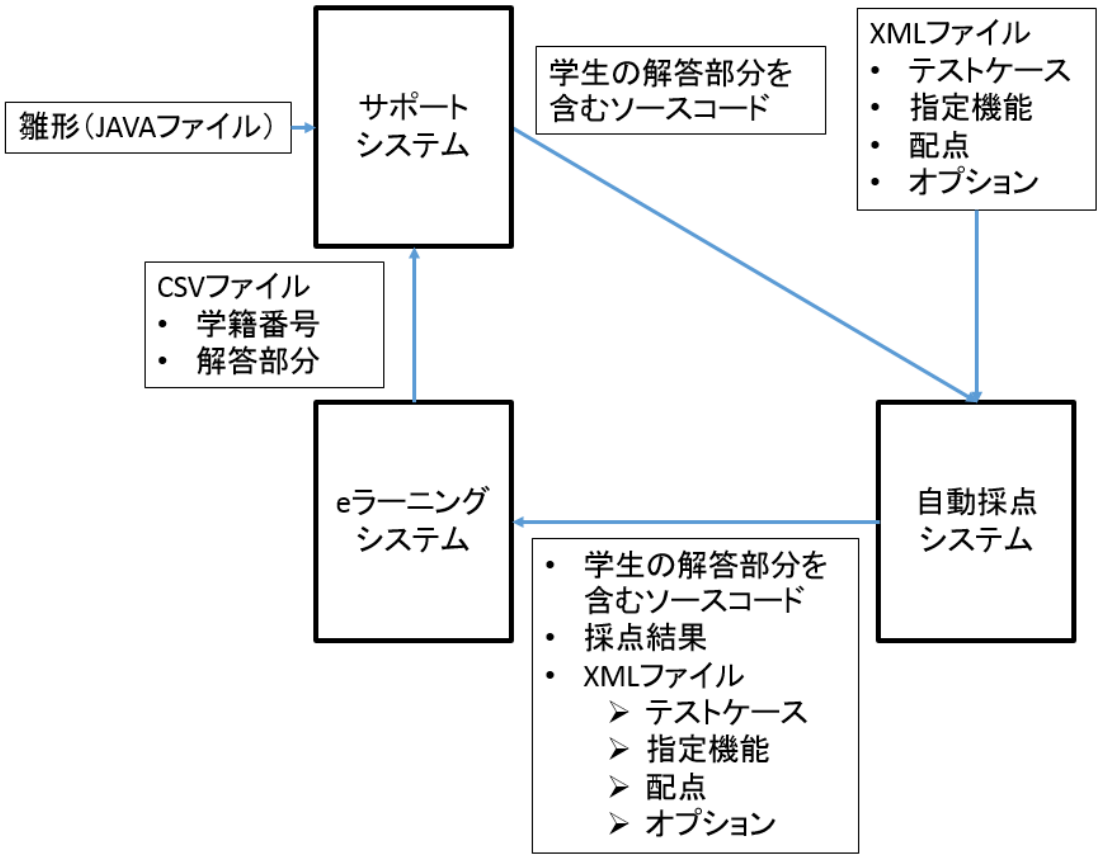


図 18　自動採点システム概念図

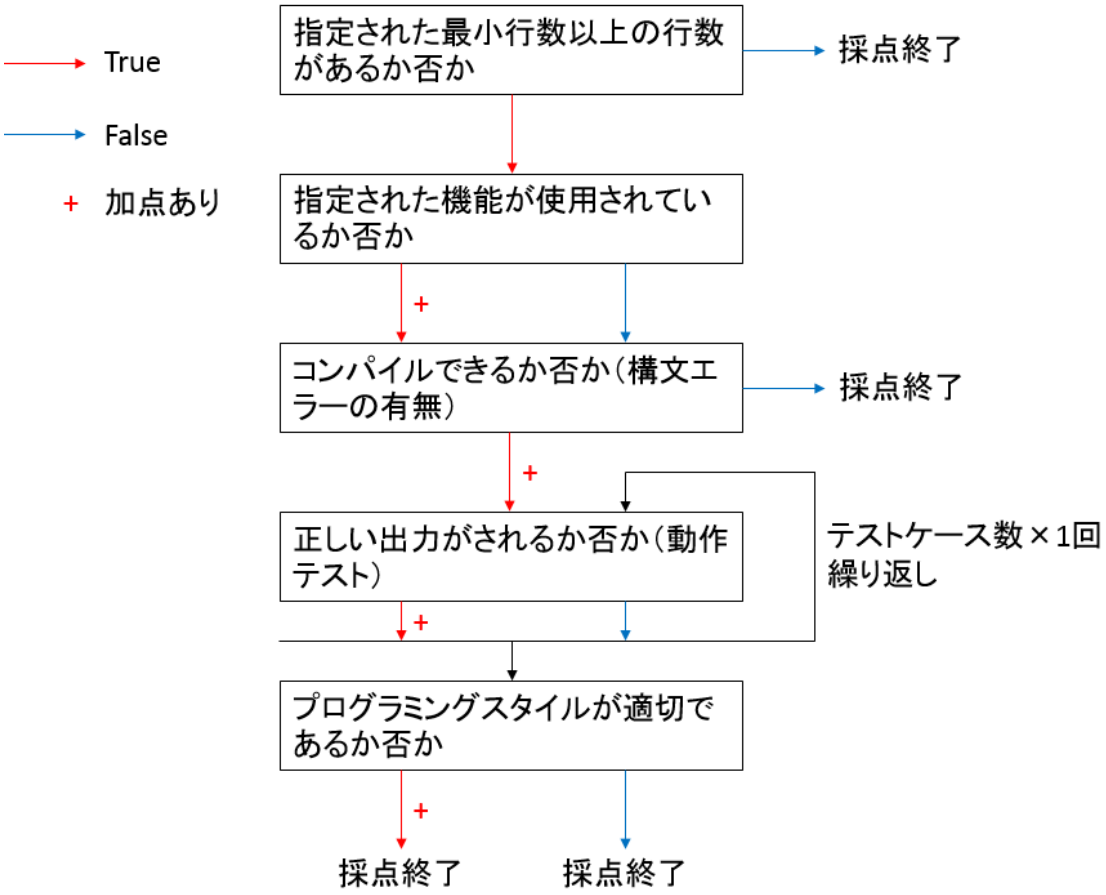
True
False
+ 加点あり
指定された最小行数以上の行数があるか否か
採点終了
指定された機能が使用されているか否か
+
コンパイルできるか否か（構文エラーの有無）
採点終了
+
正しい出力がされるか否か（動作テスト）
テストケース数×1回繰り返し
+
プログラミングスタイルが適切であるか否か
+
採点終了
採点終了

図 19　採点の流れ

# 8章　実験

7 章で提案したシステムの有用性を検証するために，いくつかの実験を行った．本章では，実験内容とその結果・考察について述べる．

- 対象学生：三重大学工学部電機電子工学科　プログラミング演習Ⅰ・Ⅱ（2015 年開講）の受講者　計 88 名
- 小テスト実施期間：2015 年 5 月 22 日～2015 年 7 月 17 日

## 8.1　実験 1：予備調査

**実験内容**：本研究で提案するシステムが完成する前に，5 章で述べた各項目の配点を考えた．図 11 のようなソースコードのコアとなる部分のみを解答させる問題を学生にいくつか解いてもらい，以下に示す項目・配点を使って手動で採点した．

- 指定機能が使用されている（10%）
  （使用機能数／指定機能数）×10%
- 構文エラーがない（30%）
  コンパイルできなければ 0%
- 正しい動作をしている（50%）
  （動作成功数／テストケース数）×50%
- プログラミングスタイルが適切である（10%）
  一箇所でも不適切なスタイルがあれば 0%

ただし，指定機能が無い場合は意図点を 0%とし，動作点を 60%とした．以降，5 章で述べた各採点項目の配点をそれぞれ，構文点，動作点，意図点，スタイル点と呼ぶ．予備調査の条件を以下に示す．

- 使用教科書：スッキリわかる Java 入門[15]
- 問題数：11 問

また，この予備調査における採点は，7 章で述べた採点とは異なる点（以降，特殊採点基準と呼ぶ）がいくつかあるため，それを以下に示す．

- プログラミングスタイルが適切であるかの判定は，コンパイルができない場合でも行った．
- 意図点の採点には，指定機能だけでなく雛形で宣言されている変数が解答部分で使われているかについても判定した．

- 例え全項目が 0 点でも，何か解答があれば 1 点を与えた．

**実験結果：**採点の結果を表 4，得点分布の一例を図 20 に示す．図 20 の得点分布は一つの問題に対するものであり，多少の違いはあるもののどの問題についても似たような形になっている．

表 4　実験 1 の結果

| 章 | キーワード | 問題 | 受験者数（名） | 平均点（%） |
|---|---|---|---|---|
| 第 2 章 | 演算 | #1 | 87 | 57 |
| | | #2 | 87 | 69 |
| | | #3 | 87 | 61 |
| 第 3 章 | 条件分岐 | #4 | 88 | 44 |
| | | #5 | 88 | 64 |
| | | #6 | 88 | 50 |
| | | #7 | 88 | 62 |
| 第 4 章 | 配列 | #8 | 88 | 26 |
| | | #9 | 88 | 35 |
| 第 5 章 | メソッド | #10 | 86 | 26 |
| | | #11 | 86 | 45 |

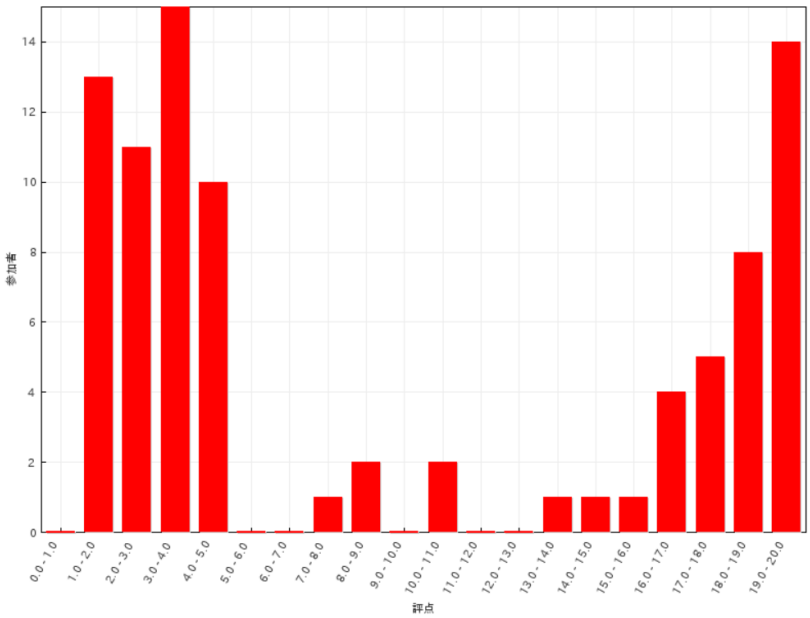


図 20　実験 1 の得点分布

**考察 :** 図 20 で示すように，満点側と 0 点側でほぼ二極化している．これでは，プログラムが動くか動かないかの判定と大差がない．これは，この実験で用いた配点では，構文エラーがあり，コンパイル不可能な場合，テストができないので動作点, も 0%となってしまうためであると考えられる. 例えば，プログラムの文末につける“;（セミコロン）”をどこか 1 箇所つけ忘れただけで，80%の減点になってしまう．プログラミングの世界はシビアであるためそれでもいいかもしれないが，初心者には厳しすぎるかもしれない.

実際にセミコロン忘れ，変数名・メソッド名などのスペルミスなど，小さなミスによって大きく点数を落としている学生が多く見られた．コードの例を図 21 に示す．セミコロンが 1 つ抜けているため，コンパイルに失敗し，ほぼ 0 点となる.

また，プログラムのコアとなる部分のみを解答させる問題ならではのミスであると思われるが，雛形で宣言されている変数を解答部分で再び宣言する二重宣言によってコンパイルエラーが出てしまう学生も多く見られた．コードの例を図 22 に示す.

上記のような要因から，表 1 を見てもわかるように，平均点が大きく下がってしまい, 講義の単位取得の基準である 60%を超える平均点が出た問題は, 11 問中 4 問しかない.

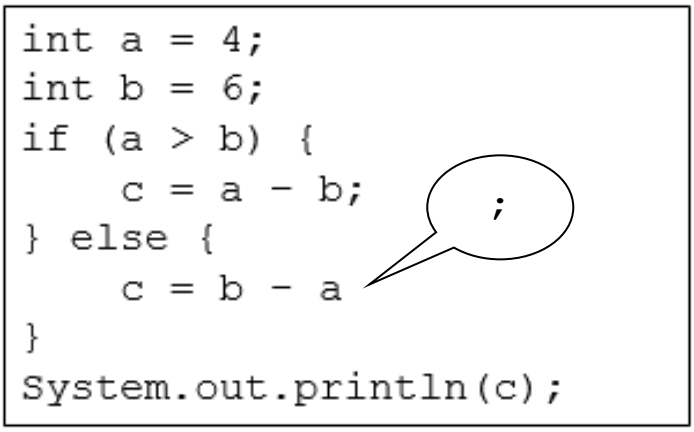


図 21　採点が厳しすぎるコードの例（一部）

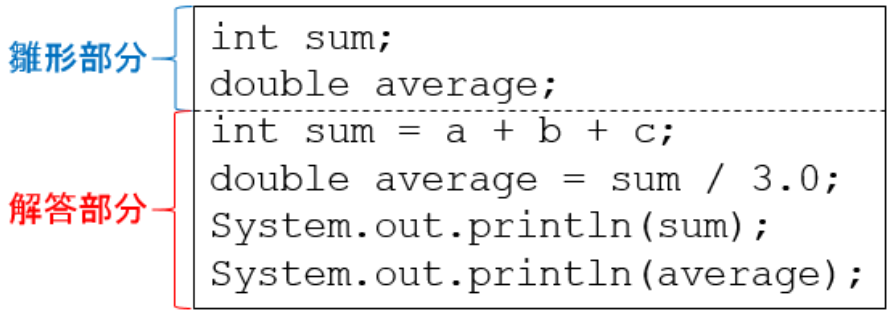


図 22　二重宣言のコードの例（一部）

逆に採点が優しすぎると考えられる解答もいくつかあった．例として，余分なコードが入っていたり，不自然な表現によって多少の動作不良があったりしても，大きな減点になっていない場合が見られた．コードの例を図 23 に示す．else の内容が無いという非常に不自然なコードであるが，コンパイルは成功し，動作テストが行われる．変数 a と b の値をキーボードから読み込む形にした場合，いくつかのテストケースでは動作も成功してしまい，大きな減点とはならないだろう．

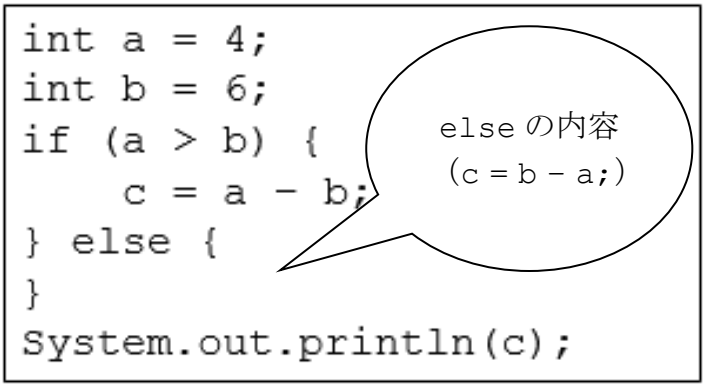


図 23　採点が優しすぎるコードの例（一部）

このように，まだまだ採点項目やその配点については課題もあり，検討の余地があると考えられるが，今回は，プログラムは動かなければ意味が無いと考え，5 章と 7 章で述べた採点基準・方法でのシステムの実装を行う．ただし，この実験で用いた配点比率は，我々が検討して出した一例であり，実際のシステムでは使用者が自由に設定できる（配点比率のみ，○×にするか部分点を与えるかなどの判定方法の変更は不可）．

### 8.2　実験 2：学生の解答時間の調査

**実験内容：**本研究では，図 11 のようなプログラムのコアとなる部分を解答させる問題形式をメインとして扱う．その理由として，2 章でも述べたように，学生のプログラミング技能を向上させるには，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ということが挙げられる．では，実際にコード全体を解答させるものと比べてどれほど時間が短縮されるのかを検証した．

コアとなる部分の問題の解答時間計測の対象としては，解答部分の想定コード行数が 2～7 行程度であり，レベルも同程度である表 3 の問題#1～#7 とする．なお，これらの問題を解答させる際の小テストの制限時間は，余裕を持って 15 分に設定した．

コード全体を解答させる問題の解答時間計測は，筆者の研究室で開発・研究を進めているプログラミング演習支援システム Propel[16]で実際にコード全体を解答させる演習課題を行っているため，その解答の編集から提出までの時間を参照した．ただし，講義＋休み時間（100 分）以内に提出できた学生のデータのみとした．また，2 分未満で提出されたデータは，他学生の解答をコピー&ペーストして提出した可能性があると考え，除外した．

**結果：**上記の 2 パターンの解答時間計測の結果を表 5 に示す．

表 5　実験 2 の結果

| 対象 | 最大 | 最小 | 平均 |
|---|---|---|---|
| コードの一部 | 15 分 | 1 分 56 秒 | 8 分 23 秒 |
| コード全体 | 96 分 | 2 分 | 32 分 25 秒 |

**考察：**表 5 で示すように，制限時間に違いがあるものの，部分的に解答させる問題の方が全体を解答させる問題よりも約 4 分の 1 の時間で解答できることがわかった．コード全体を解答させる演習も重要であるが，プログラムのコアとなる部分を解答させる小テスト形式の問題を取り入れることも有用であると考えられる．

### 8.3　実験 3：講師の採点にかかる時間の調査

**実験内容：**本研究の最大の目的は「講師の採点における負担の軽減」であるが，実際にどれ程負担が軽減できるのかを検証する．被験者に一つの問題（表 4 の問題#1）に対して，5 章で述べた採点基準，8.1 節で述べた実験 1 で用いた配点を基に手動採点をしてもらい，その採点にかかった時間（分単位）を計測してもらった．また，本研究で提案したシステムを用いて採点するにあたって必要となるファイルの作成やダウンロード・アップロードを含めた自動採点にかかる時間も計測してもらった．ただし，テストケースなどを設定する XML ファイルについては，どこに何のデータを入れれば良いかや，データ入力の形式の説明を含めた雛形を与え，テストケースなどのデータのみを編集すれば良い状態にした．実験条件は以下の通りである．

- 解答部分の想定行数：2～4 行
- 被験者数：3 名（大学院生）

**結果：**上記の時間計測の結果を表 6 に示す．

表 6　実験 3 の結果

| 被験者 | 手動採点にかかった時間（分） | 自動採点にかかった時間（分） |
|---|---|---|
| 被験者 A | 36 | 11 |
| 被験者 B | 30 | 12 |
| 被験者 C | 41 | 14 |
| 平均 | 36 | 12 |

**考察：**表 6 から，本研究で提案する自動採点システムを利用することによって，約 3 分の 1 の時間で採点ができることがわかった．表 6 を見るとそれほど大きな差ではないように思えるかもしれないが，問題数が増えれば増える程この差は大きなものとなる．これは，本研究の最大の目的である「講師の採点における負担の軽減」だけでなく，目的の 1 つである「学生に様々なプログラム作成問題を数多く解いてもらう」という観点から見ても有用であると考えられる．

### 8.4　実験 4：システムの採点の正確性の検証

**実験内容：**本研究で提案したソースコードの自動採点システムを用いて，正確に採点ができているかを検証するために，自動採点の結果と，手動採点の結果を比較した．ただし，8.1 節でも述べたように，実験 1 での手動採点では，特殊採点基準を反映させた採点となっているため，どうしても差が出てしまう．したがって，ここでの比較対象となる手動採点結果は，実験 1 の結果の特殊採点基準反映前のデータを用いる．

**結果：**上記のそれぞれの採点における各問題の平均点の比較結果を表 7 に示す．

表 7　実験 4 の結果

| 問題数 | 解答数 | 結果一致数 | 結果不一致数 |
|---|---|---|---|
| 11 | 950 | 881 | 69 |

表 7 から，自動採点システムを利用しても，概ね手動採点に近い結果が得られるが，わずかではあるがまだ誤差が生じていることがわかる．全解答の中で採点結果に誤差が生じている解答の割合は，全体の約 7.3%であった．自動採点システム側に不備があるかどうかを確認するため，誤差が生じた原因を調査したところ，69 解答全ての原因が手動採点側での見落とし・採点ミスであった．

**考察：**実験 4 において，手動採点側での採点ミスの原因を表 8 に示す．

表 8　手動採点側での採点ミスの原因

| 構文エラー | 不適切なスタイル | 動作エラー |
|---|---|---|
| 17 | 48 | 4 |

表 8 からわかるように，手動採点側での採点ミスの原因のほとんどが以下の 2 つであった．

- 構文エラーの原因の見落とし
- プログラミングスタイルが不適切な部分の見落とし

「.」や「+」などソースコード中に入り得る細かい記号のミス（「.」が「,」になっている等）や変数名のスペルミスなど，8.1 節の考察の部分で述べたよ

うな小さなミスは，手動採点においては講師にとっても見落としやすい．特に，全角スペースが入っていた場合にそれを見つけ出すことは，手動採点では非常に困難であり，先に述べた実験 4 での「構文エラーの原因の見落とし」の多くがこの全角スペースによるものであった．プログラミングスタイルについても，スペースの数など細かい部分を見なければならないため，非常に見落としやすい．

手動採点する場合でも，学生の解答のソースコードをコンパイラにかければ，上記のような見落としは軽減されるかもしれないが，いちいち全ての解答をコンパイラにかけて採点することは，非常に煩わしく，時間がかかる．

一方，本研究で提案した自動採点システムを利用することで，上記のような小さなミスを一切見逃すことなく，適切な採点ができている．採点時間の短縮だけでなく，採点精度に関しても，小さなミスの見逃しをなくすという点で有用であると考えられる．

課題としては，その採点方法の柔軟性が挙げられる．実験 4 では，手動採点において，特殊採点基準を反映させない結果を比較対象にしている．これは，同条件で採点した時にこの自動採点システムが上手く動作することを確認するためである．逆に言えば，手動であればできる特殊採点基準を反映させたような柔軟な採点ができないということである．この自動採点システムにおいても，採点方法にどこまで柔軟性を持たせることができるかが課題となると考えられる．また，手動採点における採点基準を自動採点における採点基準を統一して，システムに動作不良がないことを確認したが，採点基準を指定しなかった場合の手動採点の結果との相関を調査することも課題となる．

# 9章　まとめ

近年の情報化社会の発展に伴い，プログラミング教育の必要性が高まっており，工学系の大学や学部のみならず，文系の大学においてもプログラミングは必修科目として教育されており，学生のプログラミング技能の向上が期待される．

学生がプログラミング技能を向上させるためには，様々なプログラムを数多く作成す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という現状がある．

そこで，e ラーニングシステムを用いて，演習だけでなく小テストを取り入れることにより，学生に多くのプログラム作成問題を解いてもらう方法が考えられる．しかし，ソースコードのような表現の自由度が高い解答を自動採点することはできず，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．

本研究で提案したシステムを導入することで，講師が採点の際に，動作テストだけでなくいろいろな観点で，自動で採点することができ，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．また，これにより講師が学生に数多くの様々なプログラム作成問題を課すことができるようになり，学生のプログラミング技能が向上されることが期待できる．

実際の有用性検証実験の結果から，課題は残るものの，この自動採点システムを導入することは，講師の負担軽減や，採点の正確性に関しての効果が得られることが確認できた．

## 参考文献


[1] 「中央大学文学部 カリキュラム」, http://www.chuo-u.ac.jp/academics/faculties/letters/guide/curriculum（2016 年 1 月参照）

[2] 「日本女子大学 基礎科目詳細」, http://www.jwu.ac.jp/unv/faculty_department/curriculum_gakubu/curriculum_mejiro/kiso/（2016 年 1 月参照）

[3] Online Judge, https://github.com/hit-moodle/onlinejudge（2016 年 1 月参照）

[4] 和田修平，井上潮：“盗用発見と自動採点によるプログラミング演習課題の評価支援システム”，電子情報通信学会データ工学研究専門委員会・日本データベース学会・情報処理学会データベースシステム研究会，DEIM フォーラム 2011（2011）

[5] 北 英彦，望月 将行，高瀬 治彦，林 照峯，森田 直樹：プログラムの自動テスト機能を備えたプログラミング演習システム，日本教育工学会大会講演論文集，20 巻，pp.677-678，(2004)

[6] Moodle.org, https://moodle.org/ （2016 年 1 月参照）

[7] Blackboard：Blackboard Learn, http://www.blackboard.jp/platforms/learn/ （2016 年 1 月参照）

[8] Maplesoft，サイバネットシステム株式会社：Maple T.A., http://www.cybernet.co.jp/maple/product/maple_ta/（2016 年 1 月参照）

[9] STACK, http://stack.bham.ac.uk/（2016 年 1 月参照）

[10] 伊藤隼人，北英彦：“数式解答評価システム STACK を利用したプログラミング演習用問題”，コンピュータ利用教育議会，2014PC カンファレンス（2014）

[11] （株）永和システムマネジメント, 平鍋健児：“Java コーディング標準”, http://www.objectclub.jp/community/codingstandard/CodingStd.doc.pdf

[12] 上村拓磨，北英彦：プログラミング演習におけるコード整形ツールを用いたプログラミングスタイルのチェックに関する研究，2015 年度三重大学卒業論文（2016）

[13] Artistic Style, http://astyle.sourceforge.net/（2016 年 1 月参照）
[14] 柴田望洋：“明解 Java 入門編”，p.73，ソフトバンククリエイティブ株式会社（2007）
[15] 中山清喬，国本大悟：“スッキリわかる Java 入門”，pp.60-96，株式会社インプレス（2014）
[16] 伊冨昌幸，小島祐介，高橋功欣，北英彦：プログラムの作成情報を把握する機能を持つプログラミング演習システム, PC カンファレンス 2010（2010）

## 実績一覧

[A] 伊藤隼人，北英彦：数式解答評価システム STACK を利用したプログラミング演習用問題，PC カンファレンス 2014，2014

[B] 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, PC カンファレンス 2015, 2015

[C] Hayato Ito, Hidehiko Kita: Automatic Marking of Source Code at E-learning Systems, International Symposium for Sustainability by Engineering at MIU (IS²EMU) 2015, 2015

[D] 伊藤隼人，上村拓磨，北英彦：e ラーニングシステムにおけるソースコードを解答とする問題の自動採点，CIEC 研究会 2016，2016（採択）

## 謝辞

本論文は，筆者が三重大学大学院工学研究科博士課程前期課程に在籍中に行った研究をまとめたものである．本研究を進めるにあたり，懇切丁寧なご指導とご督励を賜った三重大学大学院工学研究科の北英彦准教授に深く感謝いたします．

また，貴重な時間をさいて本論文を査読して頂いた，三重大学大学院工学研究科の森香津夫教授，高瀬治彦准教授に深く感謝致します．

最後に，日頃熱心に討論していただいた計算機工学研究室の皆様方にお礼申し上げます．