情報処理学会論文誌　教育とコンピュータ　Vol.3　No.1　64–78 (Feb. 2017)

論　文

プログラムの誤り修正課題および正誤判定プログラムの自動生成

蜂巣 吉成[1,a)]　吉田 敦[1,b)]　阿草 清滋[1,c)]

受付日 2016年1月30日，再受付日 2016年9月30日，
採録日 2016年11月3日

概要：プログラムの誤り修正課題とその正誤判定を行うプログラムを自動生成するツールを提案する．誤り修正課題は意図的に誤りを混入させたプログラムを学習者に提示し，その誤りを正しく修正させる課題であり，デバッギングやコードリーディングの能力向上に有効である．我々はプログラムに混入させる誤りを置換，削除，挿入，移動の書き換え操作の観点から整理し，字句系列の書き換えルールとして定義した．誤り修正課題では誤りの箇所だけを編集可能にすると問題が簡単になり，編集可能箇所を多く設定すると出題者の意図しない修正などが行われることがあり，正誤判定プログラムの実現が困難になる．本研究では，正解となる編集箇所とその記述の個数の観点から誤りを整理し，編集可能箇所を設定する方法を提案する．誤りプログラムの編集可能箇所を適切に設定することで，別解を減らして学習の観点から妥当と考える解に限定し，正解字句と学習者の解答字句を比較することで正誤判定を行う．正しいプログラムと誤りを混入させるためのプログラム書き換えルール，ダミーの編集可能箇所を指定するルールから，誤り修正課題を HTML として，その正誤判定を CGI プログラムとして生成するツールを試作した．繰返しや配列，再帰関数，ポインタ，構造体，選択ソートや二分探索などの構文やアルゴリズムなどの学習に関するプログラムに対して誤り修正課題とその正誤判定プログラムが生成できることを確認した．

キーワード：プログラミング学習，誤り修正課題

A Generator for Exercises of Program Error Correction and Answer Checker Programs

Yoshinari Hachisu[1,a)]　Atsushi Yoshida[1,b)]　Kiyoshi Agusa[1,c)]

Received: January 30, 2016, Revised: September 30, 2016,
Accepted: November 3, 2016

Abstract: We propose a tool that generates exercises of program error correction, which are suitable for developing debugging and code reading skills in programming education, and answer checker programs. We define the processes of error injection as code transformation rules, and analyze types of errors from a view of rewriting operations. If learners can edit only codes which are not correct, it is easy for them to find errors. On the other hand, if they can edit any codes freely, it is difficult for grading programs to check all possible answers. We have analyzed types of errors from a view of how to correct them and adopted a strategy to restrict editable points and possible answers from the educational view. From a correct program, program transformation rules to inject errors, and rules to specify dummy editable points, our tool generates exercises as HTML files and answer checkers as CGI programs. CGI programs check correctness of answers by comparing tokens. Our tool can generate exercises that widely cover the syntax and semantics of the programming language such as loops, arrays, recursive functions, pointers, and structures, and some algorithms such as sorting and searching.

Keywords: programming education, error correction excercise

© 2017 Information Processing Society of Japan

1. はじめに

情報系の大学などにおけるプログラミング教育では，学習者は単にプログラムを作成したり，他人が書いたプログラムを読んで理解したりすること（コードリーディング）だけでなく，プログラム中のバグ（欠陥，フォールト）を発見して修正すること（デバッギング）も重要である．デバッグを行うには，意図するプログラムの理解，バグがある実際のプログラムの理解，プログラミング言語の理解，一般的なプログラムに関する専門知識などが必要であり [1]，プログラミングが得意な学習者もデバッギング能力を身につけるのは難しいという研究結果も報告されている [2]．本研究ではデバッギング能力向上のための問題作成および採点方法に着目する．プログラミング学習では，様々な問題を解いていくなかで，典型的なプログラム例などを学び，プログラミング言語の文法などに対する理解を深めていく．教える側としては，より多くの問題を出題したいが，一方で問題作成や採点のコストは削減したいという対立した要求がある．

従来の演習課題では，プログラムすべてを記述する全文記述課題や，プログラムの一部が空欄になった空欄補充課題などが主であった．全文記述課題では，課題の仕様どおりのプログラムを誤りなしで一気に完成させるのは難しいので，プログラム作成の過程では，デバッギングも体験することになる．しかし，学習者によってバグの種類や頻度が異なるので，デバッギングの学習としての効果は弱い．空欄補充課題は，コードリーディングの能力向上に役立つが，問題となるプログラム自身にはバグが含まれておらず，デバッギングには向かない．誤り修正課題は，出題者があらかじめ誤りを含むプログラムを作成し，学習者がその誤り箇所を発見し，修正する課題である（以下，本論文では「誤り修正課題」に合わせてバグ，フォールトを「誤り」と呼ぶ）．学習者は出題者が意図したデバッグ作業を体験することになるので，デバッギングの学習に効果的である．

問題作成のコストを減らすために，文献 [3], [4] などで誤り修正課題の作成を支援する方法は提案されているが，採点コストを減らすために必要な自動で正誤判定で行うシステムは少ない．これは，正解となる修正方法が複数存在する場合があり，正誤判定において正解を網羅することが難しいことが原因である．

本研究では，誤り修正課題とその正誤判定を行うプログラムを自動生成するツールを提案する．我々は誤り修正課題の作成にあたり，プログラムに混入させる誤りを置換，削除，挿入，移動の書き換え操作の観点から整理し，字句系列の書き換えルール（以下，誤り混入ルールと呼ぶ）として定義した．正しいプログラムと誤り混入ルール，ダミーの編集可能箇所を指定するルールから，誤り修正課題を HTML ファイルとして，正誤判定プログラムを CGI として生成するツールを試作した．以降，このツールを誤り修正課題生成ツールと呼ぶ．学習者は Web ブラウザで課題を表示し，解答，つまり，プログラムの修正箇所と修正後の字句を記述して Web サーバに送信する．Web サーバは CGI プログラムにより正誤判定を行って，結果を学習者の Web ブラウザに返す．

誤り修正課題では，誤りの箇所だけを編集可能にすると問題が簡単になり，プログラムの誤りではなく，編集可能箇所を探す問題になりかねない．一方で，編集可能箇所を多く設定すると，それだけプログラムを自由に記述できることになり，出題者の意図しない修正などが行われることがあり，正誤判定プログラムの実現が困難になる．問題を簡単にしすぎない，かつ，正誤判定を自動で行う，という 2 つの要求を満たすために，正解となる編集箇所とその記述の個数の観点から誤りを整理し，編集可能箇所を設定する方法を提案する．正解となる編集箇所とその記述が複数存在する誤り，つまり，別解が存在する誤りは学習の観点から妥当と考える解のみが正解となるように編集可能箇所を制限する．別解がない誤りは問題が簡単になりすぎないように誤り箇所と類似の箇所をダミーの編集可能箇所として設定する．

正誤判定プログラムは，誤りを混入させる前の字句を正解字句として，学習者の解答字句を比較し，正誤判定を行う．あらかじめ想定される正解と見なしてよい別解については，出題者が問題作成時にその字句を指定しておく．

誤り修正課題生成ツールにより，繰返しや配列，再帰関数，ポインタ，構造体，選択ソートや二分探索などの構文やアルゴリズムなどの学習に関するプログラムに対して誤り修正課題とその正誤判定プログラムが生成できることを確認した．実際に生成した問題を学習者に解答してもらい，誤り修正課題を用いた演習についての評価を行った．

本論文の構成は次のとおりである．2 章で誤り修正課題生成ツールの要件について述べる．プログラムの誤り修正課題の概略を述べ，誤りを混入させる編集操作や修正方法の観点から誤りを整理し，編集可能箇所を設定する方法や複数存在する正解の正誤判定を行う方法について説明する．3 章では，誤り修正課題生成ツールの設計と実現について述べる．誤りの混入をプログラムの書き換えルールとして記述する．4 章では，誤り混入ルールの汎用性，記述のしやすさについて評価し，生成した問題に対する出題者が意図しなかった別解について考察する．また，実際に Web による誤り修正課題を用いた演習を行い，別解の頻度，編集可能箇所の有効性，解答のインタフェースなどについて

[1] 南山大学理工学部
Faculty of Science and Engineering, Nanzan University, Nagoya, Aichi 466–8673, Japan
[a] hachisu@nanzan-u.ac.jp
[b] atsu@nanzan-u.ac.jp
[c] agusa@nanzan-u.ac.jp

ⓒ 2017 Information Processing Society of Japan

の評価を行う．5 章で関連研究と比較を行い，6 章でまとめる．なお，本論文は文献 [5], [6] での発表における議論などを参考に改訂したものである．

2. 誤り修正課題生成ツールの要件

本章では，誤り修正課題生成ツールの要件について述べる．大きく次の機能がある．

(1) 誤り修正課題の生成
- (1.1) 誤りを混入させたプログラムの作成
- (1.2) 編集可能箇所の設定
  - 問題の難易度
  - 別解の制限

(2) 正誤判定プログラムの生成
- (2.1) 誤りの修正方法
- (2.2) 別解の正誤判定

2.1 節で，誤り修正課題の概略について述べ，出題者の意図とともに誤りの例を示す．文法エラーはコンパイラにより検出できるので，本研究では文法的には正しいプログラムであるが，期待した実行結果が得られないような誤りを対象とする．

2.2 節では (1.1) について，正しいプログラムから誤りを含んだプログラムを作成するために，誤りの混入をプログラムの書き換えととらえ，編集操作に基づいて誤りを分類する．

(2) について，本研究で提案する正誤判定プログラムでは字句比較による正誤判定を採用する．学習者が修正したプログラムをコンパイルしてテストを実行し，その結果から正誤判定を行う方法も考えられるが，出題者にとって必要十分なテストケースを準備するのが負担であり，また，学習者に出題者の出題意図に合わせて修正を行わせることを重視した．

2.3 節では，正誤判定を自動で行うために，混入させた誤りを修正する方法について整理する ((2.1) の観点)．基本的には，誤りを混入させる前の字句に修正することで正解となるが，元の字句以外にも修正可能な場合があるので，正解となる修正箇所と正解となる字句の観点から整理する．

2.4 節では，(1.2) の問題の難易度の観点から，誤りではなく正しい字句ではあるが，修正が行えるダミーの編集可能箇所について述べる．(1.2) の別解を制限する観点から，誤りを混入させた箇所について出題の意図に合わない別解を防ぐための編集可能箇所について述べる．ここでの考え方に従って，出題者は編集可能箇所を設定する．

2.5 節では (2.2) について，編集可能箇所を制限しても生じる別解の正誤判定を行う方法を述べる．

なお，誤り修正課題生成ツールにより作成された課題は，自習用課題として利用されることを想定している．学習者の解答時における環境については特に制限はない．つまり，学習者はコンパイルや実行などを行わずに誤りを発見・修正してもよいし，誤りを含んだプログラムをエディタで編集してコンパイル・実行したり，デバッガを利用したりしながら解答してもよい．前者の場合はプログラムの動作を理解するためのコードリーディング能力の向上にも役立ち，後者の場合はプログラムの動作を実践的に理解するためのスキル向上に役立つ．なお，本論文で提示する課題はコードリーディングによる解答を想定し，関数を単位として提示しており，プログラムの実行に必要な main 関数を提示していない場合がある．本研究では C 言語の誤り

```
 1: #include <stdio.h>
 2: #define MAXSIZE 128
 3:
 4: int main(void)
 5: {
 6:   int data[MAXSIZE];
 7:   int size, sum, count, i;
 8:   double avr;
 9:
10:   sum = 0;
11:   size = 0;
12:   while (scanf("%d", &data[size]) != EOF) {
13:       sum += data[size];
14:       size++;
15:   }
16:
17:   avr = (double) sum / size;
18:   printf("平均: %f\n", avr);
19:
20:   count = 0;
21:   for (i = 0; i < size; i++) {
22:     if (data[i] >= avr) {
23:       count++;
24:     }
25:   }
26:   printf("平均以上は%d 個\n", count);
27:
28:   return 0;
29: }
```

図 1　配列処理の正しいプログラム
Fig. 1 Correct program of processing an array.

```
 1: #include <stdio.h>
 2: #define MAXSIZE 128
 3:
 4: int main(void)
 5: {
 6:   int data[MAXSIZE];
 7:   int size, sum, count, i;
 8:   double avr;
 9:   ____________
10:   sum = 0;
11:   size = 0;
12:   while (scan("%d",_data[size]) != EOF) {
13:       sum += data[size];
14:       size++;
15:   }
16:   ____________
17:   avr = (double) sum / size;
18:   printf("平均: %f\n",_avr);
19:   ____________
20:   for (i = 1; i <= size; i++) {
21:     if (data[i] >= avr) {
22:       count++;
23:     }
24:   }
25:   printf("平均以上は%d 個\n",_count);
26:   ____________
27:   return 0;
28: }
```

図 2　配列処理の誤りを混入させたプログラム
Fig. 2 Error program of processing an array.

© 2017 Information Processing Society of Japan

```
 1: struct person {
 2:    char name[64];
 3:    double height, weight;
 4: };
 5:
 6: void swap(double *a, double *b)
 7: {
 8:    double __tmp;
 9:    tmp = *a; *a = *b; *b = tmp;
10: }
11:
12: void sort_by_height(
13:       struct person p[], int size)
14: {
15:    int min, i, j;
16:
17:    for (i = 0; i < size-1; i++) {
18:       min = i;
19:       for (j = i+1; j < size-1; j++) {
20:          if (p[j].height <= p[min].height) {
21:             min = j;
22:          }
23:       }
24:       swap(_p[j].height, _p[min].height);
25:    }
26: }
```

図 3 構造体の配列をソートする誤りプログラム
Fig. 3 Error program of sorting an array of structure.

```
1: int strlength(char *s)
2: {
3:    if (*s == '\0') return 0;
4:    else return 1 + strlength(s-1);
5: }
```

図 4 文字列 s の長さを再帰関数で計算する誤りプログラム
Fig. 4 Error program of calculating the length of string s by recursive function.

修正課題を対象とするが，考え方は他のプログラミング言語でも適用可能である．

2.1 誤り修正課題の概略

誤り修正課題の例を図 1，図 2，図 3，図 4 に示す．図 2，図 3，図 4 における下線部は本研究で提案する編集可能箇所である（2.4 節）．図 1 は，「ファイル内のすべての整数データを配列に読み込みながら平均値を求め，配列内の平均値以上のデータの個数を表示する」正しいプログラムであり，図 2 は誤りを含んだプログラムである．図 2 では，(1) 12 行目の scanf 関数の関数名が違い，(2) 実引数にアドレス演算子&がなく，(3) 20 行目の for の初期化式と条件式が配列の添字の範囲と合っておらず，(4) 変数 count が初期化されていない．

図 3 は，名前，身長，体重からなる構造体の配列を選択ソートで身長の昇順に並べ替えるプログラムであるが，(1) 6 行目の swap 関数の引数と 8 行目の局所変数の型は正しくは struct person であり，(2) 19 行目の for の条件は j < size であり（17 行目の条件は正しい），24 行目の swap 関数の呼び出しにおいて，(3) 実引数にアドレス演算子&がなく，(4) 第 1 引数の添え字は j ではなく i であり，(5) メンバ参照 .height は不要である．

図 4 は，文字列の長さを再帰関数で計算する誤りのあるプログラムである．正しくは 4 行目の関数 strlength の再帰呼び出しの引数は s+1 である．

誤り修正課題では，プログラムに不自然な記述があると誤りの箇所が容易に推測できてしまう．また，提示されたプログラムを通して文法や典型的な処理などについて学習をしていくので，プログラムの可読性も重要である．本研究ではプログラムは次のコーディングルールに従うものとする．

(R1) 変数は 1 つの用途でしか使用しない．
(R2) 繰返し中で使われる変数は繰返しの直前で初期化する．
(R3) 意味的にまとまった処理は空行で区切る．
(R4) 連続した空行は記述しない．

出題者はこれらのルールに基づいて正しいプログラムを作成する．これにより，学習者はプログラムを入力，計算，変換，出力などの処理の集まりとして理解しやすくなる．誤り修正課題生成ツールが生成する誤りを混入させたプログラムもこれらのルールに従うようにする[*1]．課題作成において，文を削除して誤りを混入させると連続した空行が生じる場合があるが，(R4) より連続した空行は 1 行の空行になるように，誤り修正課題生成ツールはプログラムを生成する．たとえば，図 1 では，20 行目の代入文を削除すると 2 行の空行となるが，図 2 のように 1 行の空行として問題を生成する．

我々が大学で C 言語のプログラミング演習などを担当したときの経験や文献 [2], [7], [8] などを参考に誤りについて分析した（**表 1**）．誤りは，出題者が誤りの発見・修正を通して学習者に理解してもらいたい出題意図と関連づけられている．出題意図は，C 言語においてよくある誤りの発見・修正（=と==の間違いや scanf 関数の実引数の&の有無など），典型的な処理の理解（配列を先頭から最後まで走査する）などの文法などを学んでいる初学者を対象にしたものから，アルゴリズム（選択ソート）の理解など文法をひととおり学んだ後の学習まで多岐にわたる．なお，どのような誤りが学習に適しているかの議論は本論文の対象外とする．

2.2 誤りを混入させる編集操作に基づいた分類

本研究が対象とする誤り修正課題は，2.1 節で示したように，正しいプログラムに対して演算子，式，式文などの構文要素に誤りを混入させたプログラムを対象とする．プログラムの書き換えの観点から，誤りを混入させる方法を字句列の挿入，削除，置換，移動の編集操作とし，誤りを分類した（表 1）．

削除や置換は正しいプログラムから特定の字句列を指定

*1 (R2) について，繰返し中に計算される変数を繰返しの中で初期化するような誤り（図 7 (e)，図 10）はこの限りではない．

© 2017 Information Processing Society of Japan

表 1　誤りの分類
Table 1　Classification of errors.

出題意図	誤り	編集操作	正解箇所	正解記述
scanf 関数による入力	scanf 関数の実引数の&忘れ（int）（図 2（2），図 7（c））	削除	1 カ所	1 つ
scanf 関数による入力	scanf 関数の実引数の余分な&（char []）	挿入	1 カ所	1 つ
ライブラリ関数の利用	ライブラリ関数名の誤り（図 2（1））	置換	1 カ所	1 つ
ライブラリ関数の利用	関数呼び出しにおける実引数の順序の誤り	置換	1 カ所	1 つ
多分岐の条件分岐	else if における else 忘れ（図 9（e））	削除	1 カ所	1 つ
演算子	演算子の誤り（==と=, &&と||など）（図 7（a），図 9（c））	置換	1 カ所	1 つ
データ型	型の誤り（図 3（1））	置換	1 カ所	1 つ
繰返しによる配列走査	繰返しの初期値と終了条件の誤り（図 2（3））	置換	1 カ所	複数
繰返しにおける変数の初期化	変数の初期化忘れ（図 2（4））	削除	複数箇所	1 つ
繰返しにおける変数の初期化	繰返し本体で変数を初期化（図 7（e），図 10）	移動	複数箇所	1 つ
構造体とメンバ参照	不要なメンバ参照（図 3（5），図 7（d））	挿入	1 カ所	1 つ
関数の再帰呼び出し	再帰関数の実引数の誤り（図 4，図 7（b））	置換	1 カ所	複数
関数のポインタ引数	関数呼び出しにおけるポインタ引数の&忘れ（図 3（3））	削除	1 カ所	1 つ
選択ソートアルゴリズム	配列の比較範囲の誤り（図 3（2））	置換	1 カ所	1 つ
選択ソートアルゴリズム	交換する配列要素の誤り（図 3（4））	置換	1 カ所	1 つ

して，それを削除するか別の字句列に置き換えると問題が作成でき，削除，置換前の字句が正解の 1 つとなる．挿入は正しいプログラムに字句を追加することで誤りとなり，その字句を削除することで正解となる．移動は削除と挿入の組合せである．たとえば，図 1 の 20 行目の `count = 0;` を削除し，22 行目の前に挿入すると移動の誤りになる．移動は 2 つの誤りを修正することで正解となる．移動や挿入により誤りを混入させると，誤り箇所が容易に分かるような不自然なプログラムになりやすいので，置換や削除に比べると問題に適した誤りの種類は少ない．

2.3　正解となる修正方法に基づいた分類

誤りを修正するための編集箇所（正解箇所）と正解記述の個数の観点から，誤りを次の 4 通りに分類した．なお，ここでいう正解箇所が複数箇所とは，2.2 節で述べた移動の誤りのように 2 つの箇所をともに修正するという意味ではなく，2 カ所以上のいずれかで誤りを修正する記述ができるという意味である．

(a) 正解箇所が 1 カ所で，正解記述が 1 通り
(b) 正解箇所が 1 カ所で，正解記述が複数
(c) 正解箇所が複数箇所で，正解記述が 1 通り
(d) 正解箇所が複数箇所で，正解記述が複数

正解となるプログラム記述は，演算子，式，式文などになるが，式や式文については，`a+b` と `b+a` のように，表現は異なるが評価結果が同じとなる式が存在する．本研究で提案する正誤判定プログラムは，学習の観点から不自然と思われる記述（たとえば，`a+b+0` や `(a)+(b)` など）を判定対象から除外し，可換な四則演算を同一と見なして正誤判定を行う．たとえば，`a-b` と `-b+a` は同一の記述として正誤判定する．空白についても無視する．上記の分類では，これらを同じ記述と見なして正解記述を 1 通りとしている．不必要な括弧については考慮をせず，元の正しいプログラムと同じ括弧の記述を正解とする*2．

(a) は，誤りを混入させたときに，その箇所を正しいプログラムの元の字句に修正することのみが正解となる場合である．別解のない誤りである．

(b) は，誤りを設けた箇所を，元の字句以外の字句に修正しても正解となる場合である．たとえば，`n` 回繰り返す `for` 文の誤った記述である `for (i=1; i<n; i++)` は，`for (i=1; i<=n; i++)` または `for (i=0; i<n; i++)` として，複数とおりに修正できる．ただし，つねにどちらの記述でも正解となるわけではなく，`for` 文の本体に依存する．図 4 も関数の実引数を `s+1` ではなく，`++s` と修正しても正解となるが，`s+1` を `++s` としてもよいかはプログラムに依存する．つまり，この式を評価した後に `s` を参照する場合は，`++s` は不正解となる．

(c) は，正解の記述は 1 つだが，それを記述する箇所が複数ある場合であり，正しいプログラムに出現した文を削除した場合が該当する．たとえば，図 2 の変数 `count` の初期化忘れは，9 行目から 19 行目までのいずれかの行に `count = 0;` を追加すればよく，複数の箇所での修正が可能である．

(d) は，表 1 には現れていないが，誤りを混入させた箇所とは異なる箇所を修正しても正解となる場合であり，出題意図とは異なる別解が生じた場合である．たとえば，図 2 の 20 行目の `for` の初期化式と条件式の誤りについて，21 行目の配列の添字を `data[i]` から `data[i-1]` に修正しても正解となる．配列の走査として，`for (i=1; i<=size; i++)`

*2　誤り修正課題では，演算子（括弧も演算子に含む）を 2 つ以上含むような式全体を編集可能にするのではなく，演算子や被演算子に誤りを混入させて，誤りのない類似の演算子を編集可能にすると，計算において注目すべき箇所が明確になる．

© 2017 Information Processing Society of Japan

と data[i-1] の組合せは誤りではないが，典型的な配列走査の記述ではなく，不自然なプログラムであり，学習の観点からは適さない．21 行目を編集可能とせずに，編集可能箇所を 20 行目に制限すれば，この別解は生じない．つまり，編集可能箇所を適切に制限することにより，出題意図にあった修正方法に制限できる．

2.4　編集可能箇所の設定

2.3 節で分類した誤りについて，どこを編集可能箇所として設定したらよいかを考える．本節の考え方に従って，出題者は編集可能箇所を設定する．

一般に，混入させた誤りと編集可能箇所が少ないと，誤り箇所を特定しやすくなるので問題が簡単になる．逆に混入させた誤りと編集可能箇所を多く設定すると，すべての誤り箇所を特定するのが難しくなり，さらに，出題者が意図しない修正でも正解となる場合がある．(b) や (c) の誤りは，出題意図にあった修正を行うように編集可能箇所を制限すると学習効果があがると考えられる．これらを考慮して，問題を簡単にしすぎず，別解を減らすように誤りの種類ごとに編集可能箇所を検討する．意図しない別解 ((d) の場合）の自動判定は難しいので，出題者が目視で確認して別解が生じた場合は，組み合わせる編集可能箇所を変更することとする．

2.3 節の (a) は，問題が簡単になりすぎないように，誤りと構文的に類似した箇所も編集可能とする．この誤りではないが編集できる箇所をダミー編集可能箇所と呼ぶ．たとえば，scanf 関数の実引数の & 忘れの誤りでは，関数呼び出しにおいて変数を指定した実引数の直前部分をすべて編集可能とし，ライブラリ関数名の間違いではすべての関数呼び出しの関数名を編集可能とする．図 3 では，8 行目の変数 tmp の直前部分も編集可能とする．これはポインタについて深く理解していない学習者が，仮引数の *a，*b につられて *tmp に誤って修正することを想定している[*3]．図 4 の 3 行目では，return する値の 0 を編集可能としているが，これを 1 と誤って修正することを想定している．

(b) は，出題意図と別解を考慮して，編集可能箇所を設定する．2.3 節で述べた n 回繰り返す for 文の誤りでは，編集可能箇所を演算子 < に限定することで，for (i=1; i<=n; i++) のみを正解とする．

同様な for 文の誤りの例として，図 2 の 20 行目があげられる．for (i=0; i<=size-1; i++) と修正しても正解となるが，データの個数 size の減算が毎回行われており，プログラムの分かりやすさの点からは好ましくない．編集可能箇所を，初期化式では右辺のみに，条件式では演算子のみに制限することで，正解を i=0, i<size の 1 つに限定できる．この誤りは，繰返しによる配列の処理を理解するという出題意図であり，この制限により典型的な配列走査のみを正解とすることができる．

(c) は，文を削除した場合の誤りなので，修正するには文を追加する必要がある．変数の初期化忘れを典型的な誤りと考えて，本研究では文の削除は初期化文の削除を対象とし，空行を編集可能箇所とする．空行は何も記述しないままが正解の場合もあるので，ダミー編集可能箇所でもある．意図しない別解を防ぐために，学習者は空行には 1 文 (; を 1 つ) を記述することとする．図 2 では空行である 9 行目，16 行目，19 行目，26 行目が編集可能箇所となる．なお，つねに空行を編集可能にすると意図しない解答が行われる可能性があるので，文の削除の誤りが混入されているときのみ，空行を編集可能とする．たとえば，図 3 は文の削除の誤りがないので，16 行目は編集可能にはならない．

2.5　別解の正誤判定方法

2.3 節で分類した (b) における別解は，2.4 節で示した編集可能箇所を設定することにより，正解を 1 つに限定できる場合がある．我々の従来の研究では編集可能箇所を限定することで別解をなくすことを試みたが [5], [6]，実際に学習者の解答には 図 4 の ++s のように正解を限定できない場合があることが分かった．そこで，問題作成時にあらかじめ想定される正解と見なしてよい別解を出題者が指定することにする (3.3 節参照)．

(d) については，意図しない別解が生じないように出題者が編集可能箇所を設定するので，別解として扱う必要がない．

(c) の変数の初期化忘れでは，正しい文から代入文を削除することで誤りが生じ，2.4 節で示した編集可能箇所を設定することで，正解となる箇所が複数生じる．正解箇所は変数が使用されるより前に実行される箇所であり，厳密にその箇所を求めようとするならば，データ依存解析が必要になる．本研究で提案する正誤判定プログラムは，2.1 節のコーディングルール (R1)，(R2) から，削除された初期化文より前の，同じブロックに存在する空行が正解箇所となるので，データ依存解析を行わずに正誤判定を行う．図 2 では，正解プログラムにおける 20 行目より前の行である 9 行目，16 行目，19 行目のいずれかに追加した場合が正解となる．

3.　誤り修正課題生成ツールの設計と実現

2 章で整理した要件に基づいて，誤り修正課題生成ツールを設計・実現する．2 章での分析から，プログラムに混入させる誤りはプログラムの演算子，式，式文などの構文要素の書き換えルールとして記述する．ダミー編集可能箇所も同様のルールで記述する．

---

*3　4.4 節の演習では，実際にこの誤った解答があった．

© 2017 Information Processing Society of Japan

```
/***************************************
 次の
「ファイル内のすべての整数データを
  配列に読み込みながら平均値を求め，
  配列内の平均値以上のデータの個数を
  表示するプログラム」
 の誤りを訂正しなさい．
***************************************/

#include <stdio.h>
#define MAXSIZE 128

int main(void)
{
  int data[MAXSIZE];
  int size, sum, count, i;
  double avr;

  sum = 0;
  size = 0;
  while (scan("%d", data[size]) != EOF) {
    sum += data[size];
    size++;
  }

  avr = (double) sum / size;
  printf("平均: %f¥n", avr);
  for (i = 1; i <= size; i++) {
    if (data[i] >= avr) {
      count++;
    }
  }
  printf("平均以上は%d個¥n", count);

  return 0;
}
```

解答

Webブラウザによる出題画面例
(網掛け部分は編集可能箇所であり
マウスオーバした際に背景色が変わる)

編集箇所にマウスを載せる

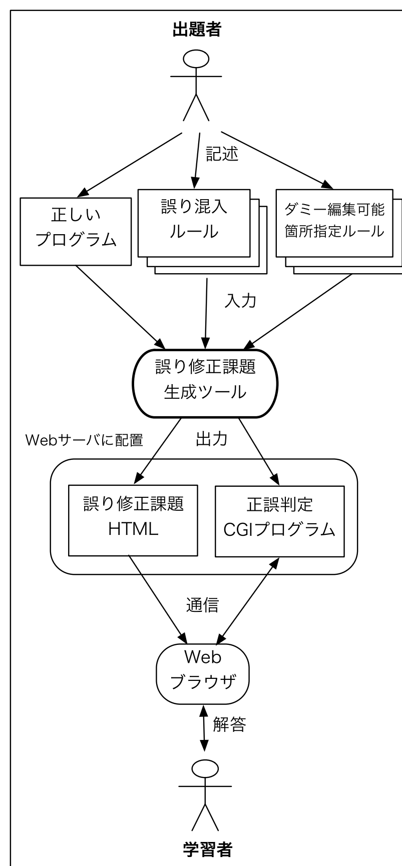
```
avr = (double) sum / size;
printf("平均: %f¥n", avr);

for (i = 1; i <= size; i++) {
  if (data[i] >= avr) {
    count++;
  }
}
printf("平均以上は%d個¥n", count);
```

クリックして編集
SAVEボタンで反映

```
avr = (double) sum / size;
printf("平均: %f¥n", avr);
count=0;
SAVE OR CANCEL
for (i = 1; i <= size; i++) {
  if (data[i] >= avr) {
    count++;
  }
}
printf("平均以上は%d個¥n", count);
```

Webブラウザによる編集操作例



誤り修正課題作成概略

図 5　誤り修正課題の画面表示・編集例と誤り修正課題作成の概略

Fig. 5　Screen shots of error correction exercises on the Web and overview of generating process.

3.1　Web による誤り修正課題

本研究の誤り修正課題は，次のようにして，Web を利用した自習用課題として利用されることを想定している．

(1) 出題者は，誤りの発見・修正を通して学習者にどのようなことを理解してもらいたいか（出題意図）を考え，正しいプログラムに対して 1 つ以上の誤りとダミー編集可能箇所が設定された誤り修正課題を誤り修正課題生成ツールにより作成する．生成された HTML ファイルと CGI プログラムを Web サーバに配置する．

(2) 学習者は Web サーバにアクセスして，誤り修正課題を解答する．学習者が誤りを修正し，「解答」ボタンを押すと CGI により正誤判定が行われ，結果が通知される．学習者は何度でも繰り返し解答できる．

学習者の Web ブラウザで誤り修正課題を表示する際には，編集可能箇所が明示されることで誤り箇所を見つけるヒントにならないように，また，修正したい箇所で直感的に編集できるように，jQuery のプラグインである JEIP を用いる*4．編集可能箇所のテキスト上にマウスカーソルを移動させるとその箇所の背景色が変わり，クリックするとブラウザ上に編集用のフィールドが表示され，SAVE ボタンを押すと，その場でブラウザ上に反映される．すべての編集を終えた後に解答ボタンを押すと，学習者が編集した箇所と編集後のテキストの情報が Web サーバ上の CGI プログラムに送信されて正誤判定が行われ，結果がブラウザ上で学習者に通知される．図 5 左に Web ブラウザでの誤り修正課題を表示した画面例を，中央に JEIP による編集画面の例を示す．問題表示画面例では，編集可能箇所を分かりやすいように網掛けで表示しているが，実際はマウスオーバした箇所のみ背景が変わる．

3.2　誤り修正課題生成ツールの入出力

図 5 右に誤り修正課題生成ツールによる問題作成処理の概略を示す．出題者は，問題の基となる正しいプログラム，正しいプログラムに誤りを混入させるための誤り混入ルール，ダミー編集可能箇所を設定するダミー編集可能箇所指定ルールを記述し，これらをツールに入力する．誤り混入ルールとダミー編集可能箇所指定ルールは 3.3 節で述べるプログラムの書き換えルールである．ツールは，これらの書き換えルールに従って，正しいプログラムを誤りが混入されたプログラムに書き換えて，誤り修正課題の HTML

*4　Joseph Scott, jQuery Edit In Place (JEIP), http://josephscott.org/code/javascript/jquery-edit-in-place

© 2017 Information Processing Society of Japan

```
double avr;
<span class="editable" id="e1">       </span>  <!-- 編集可能な空行 -->
sum = 0;
size = 0;                                               <!-- 以下の while は実際には 1 行 -->
while (<span class="editable" id="e2">scan</span>("%d",        <!-- 編集可能箇所: scan -->
          <span class="editable" id="e3"> </span>data[size])  <!-- 編集可能箇所: 実引数の前の空白 -->
          <span class="editable" id="e4">!=</span> EOF) {     <!-- 編集可能箇所: != -->
  sum += data[size];
  size++;
}
```

図 6　誤り修正課題の HTML ソースの一部

Fig. 6　A part of source HTML of an error correction exercise.

ファイルと正誤判定用の CGI プログラム（Perl）を出力する．誤り混入ルールとダミー編集可能箇所指定ルールによって指定された箇所が，誤り修正課題の HTML ファイルにおいて，プログラムを編集できる箇所となる．**図 6** は図 2 の誤り修正課題の HTML ソースの一部である．class 属性が editable の span 要素が編集可能箇所となり，一意の ID 属性が付けられる．正誤判定 CGI プログラムには，学習者が編集した箇所（ID）と修正字句が送信され，それらを元の正しいプログラムの正解字句と照合して正誤判定が行われる．

3.3　誤り混入ルール

本研究では，2 章であげたプログラムに混入する誤りをプログラムの書き換えルールとして記述する．ルールの記述において，識別子や式，文などの構文要素をパターン変数として一般化して記述することで，複数の異なるプログラムに同一の書き換えルールを適用して誤り修正課題を作成できる．プログラムの書き換えには属性付き字句系列に基づく書き換え処理系である TEBA [9] を用いる．

誤りは，正しいプログラムにおける構文要素と，誤りを混入させた後の構文要素をプログラム・パターンとして記述する（**図 7**）．正しいプログラムに対して誤りを混入させる箇所を `%before` から`%after` の間（以下，before 部と呼ぶ）に記述し，誤りを混入させたプログラムを `%after` から`%end` の間（以下，after 部と呼ぶ）に記述する．`<@` から `@>` で囲まれた部分が誤りの箇所であり，before 部におけるこの箇所の字句が正解字句となる．`${`*NAME*`:`*TYPE*`}` は，*TYPE* として記述された構文要素にマッチするパターン変数である．*TYPE* には，識別子（ID_FVAR），演算子（OPE），式（EXPR），文（STMT）などが記述できる．パターン変数により変数名や関数名，式などを抽象化することで，複数のプログラムに対して書き換えルールを適用できる（4.1 節）．図 7（a）では if の条件式の `==` を `=` に置換して誤りを混入させている．別解が想定される場合は，before 部に `@` で区切って複数の字句を記述すると，正誤判定プログラムがそれらも正解として判定する．図 7（b）は図 4 の誤りと別解の例である．削除による誤りは，after 部において削除する箇所を `<@ @>` の空欄にし（図 7（c）），

```
// 演算子の誤り
%before
if (${e1:EXPR} <@ == @> ${e2:EXPR})  ${s:STMT}
%after
if (${e1} <@ = @> ${e2}) ${s}
%end
```

（a）置換の誤り混入ルール例

```
// 関数の再帰呼出しの誤り （図 4）
%before
return ${e:EXPR} ${op:OPE}
  ${f:ID_FVAR}(<@ ${s:ID_FVAR} + 1 @ ++${s} @>) ;
%after
return ${e} ${op} ${f}(<@ ${s} - 1 @>);
%end
```

（b）別解のある置換の誤り混入ルール例

```
// scanf 関数の実引数の&忘れ （図 2（2））
%before
scanf(${e:EXPR},<@ & @>${v:ID_FVAR});
%after
scanf(${e}, <@ @>${v});
%end
```

（c）削除の誤り混入ルール例

```
// 構造体の不要なメンバ参照の追加 （図 3（5））
%before
swap(${a1:EXPR}<@ @>,${a2:EXPR}<@ @>);
%after
swap(${a1}<@ .height @>, ${a2}<@ .height @>);
%end
```

（d）挿入の誤り混入ルール例

```
//  for 文の直前の初期化文の移動 （図 10）
% before
<@ ${var:ID_FVAR} = ${lit:LITERAL}; @>
for (${e1:EXPR}; ${e2:EXPR}; ${e3:EXPR}) {
    <@ @>
    ${s:STMT*} $;
}
% after
<@ @>
for (${e1}; ${e2}; ${e3}) {
    <@ ${var} = ${lit}; @>
    ${s} $;
}
% end
```

（e）移動の誤り混入ルール例

図 7　誤り混入ルールの記述例

Fig. 7　Samples of error injection rules.

挿入による誤りは，before 部において挿入される箇所を `<@ @>` の空欄にする（図 7（d））．移動による誤りは，削除と挿入を組み合わせて記述する（図 7（e））．

3.4　ダミー編集可能箇所指定ルール

問題を簡単しすぎないためのダミーの編集可能箇所も，

© 2017 Information Processing Society of Japan

```
// printf の関数名と第 2 引数の前を
// ダミー編集可能箇所にする
% before
<@ printf @>(${e1:EXPR},<@ @>${e2:EXPR})
% after
<@ printf @>(${e1},<@ @>${e2})
% end
```

図 8　ダミー編集可能箇所指定ルールの記述例
Fig. 8　A sample of a dummy editing place rule.

誤り混入ルールと同様の書き換えルールで記述する．ただし，変換後も変換前の正しいプログラムと同じになるので，書き換えルールの before 部と after 部がパターン変数の*:TYPE* 以外は同じ記述となる．`<@ @>` の箇所がダミー編集可能箇所となる（図 8）．

3.5　実現

誤り修正課題生成ツール（図 5 右）を Perl とシェルスクリプトを用いて試作した．C プログラムの構文解析やプログラムの書き換えには TEBA [9] を利用した．ツールの規模は Perl 約 250 行，シェルスクリプト約 100 行で，生成される正誤判定 CGI プログラムは約 100 行である．

誤り混入ルールを 36 種類（置換 24，削除 8，挿入 2，移動 2），ダミー編集可能箇所指定ルールを 14 種類記述し，繰返しや配列に関する基本的な問題（図 2）や再帰関数（図 4），文字列処理，ファイル処理，二分探索，構造体配列の選択ソート（図 3）などの 10 のプログラムに適用し，誤り修正課題と正誤判定プログラムが生成できることを確認した[*5]．

4.　評価

誤り修正課題を作成するためのプログラムの書き換えルールについて，汎用性と書きやすさの観点から 4.1 節と 4.2 節で評価を行う．学習者が誤り修正課題を解答したときに実際にあった別解について 4.3 節で述べる．4.4 節では，Web による誤り修正課題を用いた演習について総合的に評価を行う．

4.1　書き換えルールの汎用性

誤り混入ルールやダミー編集可能箇所指定ルールの汎用性について議論する．一度記述したルールが複数の異なるプログラムに対して適用できれば，問題作成の手間が軽減できる．一般に，プログラミング言語の文法や典型的な処理などに関する誤り混入ルール（図 7 (a)，(b)，(c)，(e)）は汎用性が高く複数のプログラムに対して適用でき，特定のデータ構造やアルゴリズムなどについての誤り混入ルール（図 7 (d)）は適用可能なプログラムが限定されることが多い．

今回記述した 50 個の書き換えルールのうち，複数のプログラムに適用可能なルールは 35 個であり，70%は再利用可能である．これらは，`=` と `==` の間違いや，変数の初期化忘れなどの一般的な C プログラムにおけるよくある誤りが該当する．残りの 30%は構造体におけるメンバ名の誤りや二分探索関数における探索範囲の誤りなど，データ構造やアルゴリズムを理解していないときに生じやすい誤りである．

また，正しいプログラムに対して異なった誤り混入ルールを適用することで，1 つのプログラムから複数の誤り修正課題が作成可能である．たとえば，図 3 では，24 行目で不要なメンバ参照をしているが，20 行目の `.height` を削除して必要なメンバ参照のない誤り修正課題を作成することもできる．図 4 では，再帰関数の帰納段階に誤りがあるが，`if (*s=='\0') return 1;` として基底段階に誤りを含む問題も作成できる．指導者が学習者に何を学習させたいかによって，誤り混入ルールやダミー編集可能箇所指定ルールを選択して問題を作成することができる．

4.2　誤り修正課題の作成

本学でプログラミングの授業を担当している教員 2 名に誤り混入ルールとダミー編集可能箇所指定ルールを記述してもらい，これらのルールの記述のしやすさについて評価した．なお，2 名とも TEBA の概略，つまり，C 言語のプログラムを解析して，プログラムの変換ができることについては知っているが，書き換えルールの書き方などの詳細な知識はない．

どのような誤り修正課題を作成するかが決まっている状況で，ルールが記述できるかについて評価を行うために，次のような手順で教員によるルール作成を行った．各問について 30 分から 40 分ほどでルールは記述できたが，後述するように複数のルールで同じ箇所を書き換えて，正誤判定が正しく行えない例があった．なお，作成したルールは各問について 5 つであった．

(0) 教員が作成する誤り修正課題は累乗を計算するプログラムと文字列を変換するプログラムの 2 問である．累乗の課題の誤りは置換の編集操作で混入される誤り，文字列変換の誤りは置換，削除，挿入の編集操作で混入される誤りである．課題の仕様，つまり，正しいプログラムとツールに生成させる誤りを含んだプログラム，混入する誤りとダミーの編集可能箇所の一覧を事前に筆者が作成した（付録 A.1）．
Web サーバや誤り修正課題生成ツールは事前に筆者が設定し，教員はルールが記述されたファイルを作成して，コマンドを実行すると，誤り修正課題の HTML ファイル，正誤判定 CGI プログラムが生成されて，Web サーバに配置されるようにした．

(1) 筆者が Web による誤り修正課題，誤り修正課題生成ツールの使い方，誤り混入ルールとダミー編集可能箇所指定ルールの記述方法について 30 分ほど教員に説

[*5] http://ecq.tebasaki.jp/で公開している．

© 2017 Information Processing Society of Japan

```
%before
 scanf(<@"%lf"@>, &${v:ID_FVAR});
%after
 scanf(<@"%d"@>, &${v});
%end
```

（a） scanf 関数の double の書式指定子の誤り混入ルール 1

```
% before
scanf(<@ "%lf" @>, &x) ;
% after
scanf(<@ "%d" @>, &x) ;
% end
```

（b） scanf 関数の double の書式指定子の誤り混入ルール 2

```
%before
if (${e1:EXPR} <@ && @> ${e2:EXPR}) ${s:STMT*} $;
%after
if (${e1} <@ || @> ${e2}) ${s} $;
%end%end
```

（c） if の条件式の && を || にする誤り混入ルール

```
% before
<@ else @> if ('A' <= *s <@ || @> *s <= 'Z')
    ${s:STMT*} $;
% after
<@      @> if ('A' <= *s <@ || @> *s <= 'Z')
    ${s} $;
% end
```

（d） 文字列変換における誤り混入ルール（else の削除）

```
%before
if (${e1:EXPR}) ${s1:STMT*} $;
<@ else @>if (${e2:EXPR}) ${s2:STMT*} $;
%after
if (${e1})  ${s1} $;
<@  @>if (${e2})  ${s2} $;
%end
```

（e） if - else if の else を削除する誤り混入ルール

図 9　ルール記述の評価実験における誤り混入ルールの例

Fig. 9　Samples of error injection rules in evaluation experiments.

```
sum = 0;
for (i = 0; i < size; i++) {
    sum = sum + data[i];
}
```

正しいプログラム

```

for (i = 0; i < size; i++) {
    sum = 0;
    sum = sum + data[i];
}
```

誤りを含んだプログラム

```

for (i = 0; i < size; i++) {
    if (i==0) sum = 0;
    sum = sum + data[i];
}
```

意図しない解答

図 10　移動の誤りにおける意図しない解答例

Fig. 10　Unexpected answer on moving statement.

明した．説明には 図 2，**図 10** などの課題を用いて，それらの課題のルールを例として示した．

(2) 付録 A.1 の課題の仕様を教員に提示し，各教員はそれぞれルールを記述して，誤り修正課題を作成した．

図 9 の (a)，(b)，(c)，(d) が実際に教員が作成したルールである．scanf 関数の double の書式指定子を"%lf" から "%d" にする誤りはパターン変数を用いて汎用的に記述したルール（a）と，対象プログラムの変数名などを直接指定したルール（b）があった．(c) はパターン変数を用いて汎用的に記述されている．(d) は（c）のルールを適用して && を || に書き換えた後に適用するために，元の正しいプログラムにはない || をダミー編集可能箇所として指定している．この場合，生成された正誤判定プログラムは正しい正誤判定ができなかった（||を正解として判定する）．これは（e）のように条件式全体をパターン変数にすれば解決する．ルール（a）についても，このルールを適用した後で scanf 関数の int の書式指定子 "%d" をダミー編集可能箇所にするルールを適用すると，混入させた誤りの "%d" にマッチして正しい正誤判定が行えなかった．これはルールを適用する順序を逆にすれば正しく課題を生成できる．

ルールを 1 つ記述することには大きな支障はないが，再利用可能な汎用的なルール記述や，複数のルールの組み合わせ方にはノウハウが必要となる．特に，複数のルールを適用する場合は，一度書き換えた箇所にマッチしたルールはツールが警告を表示するなどの対応を検討している．

教員からの意見として次のようなものがあった．

- 慣れるのに少々時間がかかるが，慣れれば単純なパターンは書きやすい．
- 1 行に複数の誤りが含まれる場合などは，記述に少し手間がかかる．
- 誤りを混入させた箇所をさらに書き換えると期待した結果にならない．
- すでに書かれたパターンの例などを参考にすると書きやすい．
- パターンを記述するには，式や文などの文法の知識が必要になる．
- ダミー編集可能箇所を設定する際は，before 部だけ書けばよいのではないか．

ダミー編集可能箇所はルールにおいて，before 部と after 部で同じ記述になるので，before 部の記述から after 部を自動で生成する方法を検討中である．

4.3 出題者の意図しない別解の評価

生成した誤り修正課題に意図しない別解が生じないかを確認するために，学部 3，4 年生（C 言語のプログラミングは 1，2 年次に学習済み）に実際に解答してもらったところ，いくつかの問題で冗長な解答が見つかった[*6]．

図 2 において 9 行目に avr=0; を記述するような冗長な解答があった．avr は 17 行目で代入されるので，初期化の必要はない．図 3 でも 17 行目の for の条件を i < size にする冗長な条件の解答があった．配列の総和を求めるプ

*6　既発表の文献 [5], [6] で行った演習と 4.4 節で述べる本論文執筆のために行った演習で見つかった解答である．

© 2017 Information Processing Society of Japan

Q1　プログラミングは （1）得意 （2） 普通 （3） 苦手
Q2　問題を解答したことにより，プログラムに対して
　（1） 理解が深まった（理解が曖昧な箇所がわかった）
　（2） 理解は解答前と変わらない
　（3） より分からなくなった
Q3　修正したいけど修正できなかった箇所が
　（1） たくさんあった（2） 少しあった （3） なかった
Q4　問題の解答のインタフェースは
　（1） わかりやすい （2） 普通 （3） わかりにくい

図 11　アンケート設問
Fig. 11　Questionnaires.

ログラムにおいて，総和の初期化文 `sum=0;` を繰返し内に移動した問題では，繰返しの回数を条件判定する解答もあった（図 10）．値を返さない関数の最後が編集可能な空行のときに `return;` を追加する解答もあった（付録 A.2，空行のままが意図した解答）．

これらを正解とするか不正解とするかは出題者の意図に依存する．必要のない文や条件を記述するということは制御やデータの流れを十分に理解していないと考えられるので，我々は学習の観点からは不正解として扱ってもよいと考えるが，これらを正解としたい場合は誤り混入ルール作成時に別解として記述することになる．今回の提案ツールでは，置換における誤りにおいて冗長な条件などは別解として指定できるが，正誤判定においてデータフロー解析などは行っていないので，冗長な代入文を正解として扱うことはできない．

4.4　Web による誤り修正課題の演習

4.4.1　概略

Web による誤り修正課題を演習として行った際に，学習者の別解はどの程度の頻度で起こるのか，学習者のプログラム理解にどのような効果があるのか，学習者が修正したい箇所は出題者が設定した編集可能箇所と一致しているか，解答のインタフェースが使いやすいかの確認を行うために，学部 3 年生 16 名（C 言語のプログラミングは 1，2 年次に学習済み）に誤り修正課題を約 90 分間で 10 問出題した．プログラムのコンパイル，実行はせずにプログラムを読んで，解答をしてもらった．教員は教室にはいたが，解答中にヒントを出したり，正解を示したりはせずに自習形式で演習を行った．各問題の正解数や図 **11** のアンケートに対する回答を紙面に記入して提出してもらった．

4.4.2　想定される誤った解答や別解

2.4 節や 4.3 節で述べた想定される誤った解答や別解が，今回の演習でどの程度行われたのかを調べた結果を表 **2** に示す．今回生成した正誤判定 CGI プログラムでは，個人を判別できるような形式でログを記録しておらず，解答人数が分からなかったので，各問題の学生からの解答送信回数と，その中で該当解答が行われた回数とその割合を調べた．

表 2　想定される誤った解答や別解の解答数
Table 2　The numbers of expected errors and alternative answers.

問題	解答	解答回数	解答送信回数	割合
図 2	9 行目 `count=0;` 追加	21	81	25.9%
	16 行目 `count=0;` 追加	0	81	0%
	19 行目 `count=0;` 追加	5	81	6.2%
	9, 16, 19 行目 `avr=0;` 追加	8	81	9.8%
	9, 16, 19 行目 `i=0;` 追加	7	81	8.6%
図 3	17 行目 `i<size` に修正	12	302	3.9%
	8 行目 `tmp` の前に `*` 追加	11	302	3.6%
図 4 改	`return 1;` のまま	29	186	15.5%
	`strlength(++s)` に修正	0	186	0%
図 A·5	10 行目 `return;` 追加	2	126	1.6%

表 3　アンケートの回答結果
Table 3　Results of questionnaires.

問	（1）	（2）	（3）
Q1	**1** (87.7%, -)	**7** (71.3%, 19.4)	**8** (67.9%, 10.5)
Q2	**6** (78.6%, 12.3)	**8** (71.0%, 10.5)	**1** (30.9%, -)
Q3	**5** (54.5%, 12.7)	**7** (77.6%, 8.7)	**3** (74.9%, 13.2)
Q4	**5** (70.6%, 12.1)	**9** (70.5%, 18.6)	**2** (71.0%, 4.3)

図 2 では，26 行目に文を追加した解答はなかった．図 4 は，今回の演習では 3 行目を `if (*s != '\0') return 1;` として出題した．`return 1;` の解答は，誤りをそのまま修正しなかった解答である．別解として想定した `++s` の解答は今回の演習ではなかったが[*7]，以前に行った入学年度の異なる学習者を対象にした演習では，482 回の解答送信回数中 14 回（2.9%）あった．割合としてはそれほど高くないが，これらの解答が実際に行われることが確認できた．

図 2 で 9 行目に `count=0;` を追加した解答が多かったのは，10 行目の `sum = 0;`，11 行目の `size = 0;` と初期化の代入文が連続していることが理由として推測される．図 2 では，9 行目に `i=0;`，16 行目に `avg=0;`，19 行目に `count=0;` を追加した解答が 4 回あった（同一人物と思われる）．図 10 の意図しない解答は今回の演習ではなかった．図 3 の 17 行目を `i<size-2` とした解答は 5 回（1.6%）あった．プログラムのロジックを深く考えず，数値を変えて解答を送信してみて，正解となるか試したと考えられる．別解や誤った解答などを調べることで，学習者がどのようにプログラムを理解して，修正していくかを分析できる可能性がある．

4.4.3　アンケート

図 11 のアンケートの結果を表 **3** に示す．太字が回答者数で，括弧の中はその回答をした学習者の平均正答率と標準偏差である．各学習者の正答率は (全問題の正解箇所の合計 / 全問題の編集可能箇所の合計) × 100 として計算

*7　正しくない `s++` は 2 回，`*s++` は 3 回あった．

© 2017 Information Processing Society of Japan

表 4　Q1 と Q3 のクロス集計
Table 4　Cross tabulation of Q1 and Q3.

		Q3		
		たくさんあった	少しあった	なかった
Q1	得意	0	0	1
	普通	1	3	2
	苦手	4	4	0

した*8．全員の平均正答率は 70.6%，標準偏差は 15.6 である．**表 4** は Q1 と Q3 のクロス集計の結果である．

Q2 は，解答前と理解が変わらないが半数となった（1 名は未回答）．自由記述の感想・意見としては次のものがあった．

- 理解が深まったと回答した学生
  - char 型の部分が苦手なのがよく分かった．
  - 文字列が理解できていなかったと感じた．
  - 自分がどのような箇所に苦手意識を持っているかが分かった．
- 理解は解答前と変わらないと回答した学生
  - 理解していないのが分かった．
  - 理解しているかしていないかの確認はできたが，理解は深まっていない．
  - 1，2 年次の講義でできていた内容のことが分からなくなっていて復習が必要だと感じた．
- より分からなくなったと回答した学生
  - 具体的にどう違うのかやどこを参考にすると良いということを表示してほしい．

曖昧な知識ではプログラムを読んで誤りを発見，修正することは難しく，学生には自身の理解が不十分な箇所が明確になったといえる．

今回の演習では正誤判定を繰り返し行えるが，最後に正解を提示することはしていないので，全問正解しなかった学生は何が誤りだったか分からないままの状態である．文献 [4] では，間違い探し形式の演習課題では採点結果のフィードバックを行って学習者の理解度に対する認識を確実にするプロセスが重要であると指摘している．今後の課題として，一定時間内に全問正解しなかった場合は，ヒントを出したり，正解と解説を表示したりするなどの機能を検討していく．

Q3 については，80%の学生が設定された編集可能箇所以外を修正したかったと回答している（Q1 でプログラミングは普通と回答した学習者のうち 1 名は Q3 を未回答）．表 4 より，プログラミングが苦手と回答した学習者の方が，修正したいができなかった箇所が多く，表 3 より，「多かった」と回答した学習者の正答率が低い．これは，プログラミングが苦手な学習者ほど教育者が意図していない解答をする可能性があることを示唆している．編集可能箇所を制限することで，そのような教育者の意図に合わない解答を減らすことができたともいえる．学習者に誤り修正課題を自由記述で修正させてプログラミングの習熟度と修正方法との関連を調べ，実際の学習者の視点から編集可能箇所を設定する方法について，今後検討したい．

Q4 のインタフェースについては分かりやすいと普通が 9 割弱なので，大きな問題はないといえる．

*8　正答率は学習者が紙面に記入した正解数を用いて計算した．

5.　関連研究

AEGIS は，XML で記述された文書から選択肢形式，空欄補充形式，誤り修正形式の練習問題を生成するシステムである [3]．誤り修正課題は選択肢問題の誤選択肢を誤りとして混入して作成される．しかし，問題ごとに XML 文書を記述する必要があり，作業の自動化は行われていない．誤り修正課題の解答方法は，フォームのテキストフィールドに問題文中の誤り箇所の番号と修正後の字句を記入する方法である．

本研究では，誤りやダミー編集可能箇所をプログラムの書き換えルールとして記述しており，再利用可能である．出題者の意図に適した書き換えルールがすでに記述されていれば，出題者は正しいプログラムを用意して，書き換えルールを適用することで問題を作成できる．課題の解答では，JEIP を用いることで Web ブラウザに表示されたテキストをクリックして，その場で直感的に編集できる．

文献 [4] は，アルゴリズム学習のために，アルゴリズムを分割統治法や動的計画法などのパラダイムに分けて誤り混入箇所を決め，構文的に書き換えて，誤り混入プログラムを生成する方法を提案している．正しいプログラムとそのアルゴリズムの種類，誤りの数，生成するファイル数を指定すると，自動的に誤りプログラムを生成するシステムを提案しているが，自動で正誤判定を行う方法は提案されていない．

本研究では，誤りをプログラムの書き換えルールとして記述して，出題者がルールを選択することで，アルゴリズム学習に限らず，文法学習のための誤り修正課題を自動生成することができ，さらに正誤判定を行うプログラムを生成する方法を提案している．文献 [4] では，自動で挿入する誤りとして，演算子の置換（+を-に置換など）や，二項演算の一方の項を削除，関数呼び出し文の削除，制御文の条件式の削除（`if (S) A else B` を `A` に変更）などがあげられている．さらに，自動生成は難しいが学習に有効な誤りの例として，純変数から配列変数への置換（二分探索において，`x < a[mid]` から `a[right] < a[mid]` など）がある．これらは本研究で提案する書き換えルールでも手動ではあるが記述することができ，正誤判定も元の字句どおりのみを正解とするならば可能である．しかし，関数呼び出し文の削除や制御文の条件式の削除は元の字句以外にも修正方法が存在すると考えられるので（たとえば，

© 2017 Information Processing Society of Japan

`if (!S) B else A` など），別解も含めた正誤判定の方法については今後の課題である．

プログラムに故意に微少な変化（誤り）を加えて，ミュータントと呼ばれるプログラムを生成し，テストセットがその変更点を検知できるかを評価するミューテーション法が知られている [10], [11]．文献 [11] は，C 言語に対してミュータントを生成する規則（ミュータントオペレータ）を，文，演算子，変数，定数の観点から整理・分類している．たとえば，文の削除や 2 項演算子，変数名，リテラルの置換などのミュータントオペレータを定義している．

誤りを混入させたプログラムを作成するという点では，ミュータント法と本研究は似ているが，ミュータントプログラムを利用する目的が異なる．ミュータント法ではテストセットの誤り検出の評価に用いるので，ミュータントプログラムの可読性などは考慮されない．たとえば，図 1 に文を削除するオペレータを適用し，23 行目の `count++;` を削除して，then 部が空の `if` 文を作成することもある．しかし，一般的に，then 部が空の `if` 文は不自然であり，誤り修正課題としては適切ではない．

本研究では，誤りを混入させたプログラムを学習者が読んで，誤り箇所を発見・修正させることを目的にしている．混入させる誤りは，出題意図に合い，実際に学習者がおかしがちであること，および，誤りが混入されたプログラムを学習者が読んで不自然ではないことが求められる．換言すると，本研究では誤り修正課題に適したミュータントプログラムの作成方法を提案しているともいえる．

6. おわりに

本研究では，正しいプログラムに対し，誤り混入ルール，ダミー編集箇所設定ルールを適用して，誤り修正課題と正誤判定を行うプログラムを生成するツールを試作した．教員にルールを記述をしてもらい，記述のしやすさについて評価した．実際に生成した問題を学習者に解答してもらい，Web による誤り修正課題演習の評価を行った．

今後の課題としては，多くの問題を作成し，多くの学習者に解答してもらい，解答データを収集して別解や編集箇所を分析することや全問正解しなかった学習者に対する理解支援，問題作成のための書き換えルールの選択を支援する方法の考察などがあげられる．文献 [4] や本研究などを契機として，どのような誤りがプログラミング学習に効果的であるかなどの議論が活発になることを期待する．

謝辞　課題作成にご協力くださった先生方，誤り修正課題の演習に参加してくれた学生の皆様，有益なコメントをくださった査読者の皆様に感謝いたします．本研究の一部は，JSPS 科研費 26350344，2015 年度，2016 年度南山大学パッヘ奨励金 I-A-2 の助成を受けて実施した．

参考文献

[1] Ducassé, M. and Emde, A.-M.: A Review of Automated Debugging Systems: Knowledge, Strategies and Techniques, *Proc. 10th International Conference on Software Engineering*, *ICSE '88*, Los Alamitos, CA, USA, pp.162–171, IEEE Computer Society Press (1988).

[2] Ahmadzadeh, M., Elliman, D. and Higgins, C.: An Analysis of Patterns of Debugging Among Novice Computer Science Students, *SIGCSE Bull.*, Vol.37, No.3, pp.84–88 (2005).

[3] 菅沼　明，峯　恒憲，正代隆義：学生の理解度と問題の難易度を動的に評価する練習問題自動生成システム，情報処理学会論文誌，Vol.46, No.7, pp.1810–1818 (2005).

[4] 長瀧寛之，伊藤亮太，大下福仁，角川裕次，増澤利光：アルゴリズム学習における間違い探し形式の演習課題を自動生成する手法の提案と評価，情報処理学会論文誌，Vol.49, No.10, pp.3366–3376 (2008).

[5] 蜂巣吉成，吉田　敦：プログラミング初学者向けの誤り訂正問題の生成方法の提案，ソフトウェア工学の基礎 XX (FOSE 2013)，pp.35–40 (2013).

[6] Hachisu, Y. and Yoshida, A.: A Web-Based System for Error Correction Questions in Programming Exercise, *Artificial Intelligence Technologies and the Evolution of Web 3.0*, pp.124–143, IGI Global (2015).

[7] Hristova, M., Misra, A. et al: Identifying and correcting Java programming errors for introductory computer science students, *SIGCSE '03*, ACM, pp.153–156 (2003).

[8] Jackson, J., Cobb, M. and Carver, C.: Identifying Top Java Errors for Novice Programmers, *Frontiers in Education, 2005. FIE '05. Proc. 35th Annual Conference*, pp.T4C–T4C (2005).

[9] 吉田　敦，蜂巣吉成，沢田篤史，張　漢明，野呂昌満：属性付き字句系列に基づくソースコード書き換え支援環境，情報処理学会論文誌，Vol.53, No.7, pp.1832–1849 (2012).

[10] Jia, Y. and Harman, M.: An Analysis and Survey of the Development of Mutation Testing, *IEEE Trans. Softw. Eng.*, Vol.37, No.5, pp.649–678 (2011).

[11] Agrawal, H., DeMillo, R.A. et al.: Design of Mutant Operators for the C Programming Language, Technical Report SERC-TR41-P (1989).

付　　録

A.1 誤り修正課題の仕様

A.1.1 累乗

実数 x と非負整数 n を入力し，x^n を計算するプログラム．

- 誤り
  - `scanf` 関数の `double` の書式指定子を `"%lf"` から `"%d"` にする．
  - 累乗の初期値を `1` から `0` にする．
  - 累乗の計算 `pow = pow * x;` を `pow = pow * n;` にする．
- ダミーの編集可能箇所
  - `scanf` 関数の `int` の書式指定子 `"%d"`
  - `for` の繰返しの初期値 `0`

A.1.2 文字列の変換

文字列を入力し，大文字と小文字を相互変換した文字列

© 2017 Information Processing Society of Japan

```
 1: #include <stdio.h>
 2:
 3: int main(void)
 4: {
 5:   double pow, x;
 6:   int n, i;
 7:
 8:   printf("x? ");
 9:   scanf("%lf", &x);
10:   printf("n? ");
11:   scanf("%d", &n);
12:
13:   pow = 1;
14:   for (i = 0; i < n; i++) {
15:     pow = pow * x;
16:   }
17:
18:   printf("%f\n", pow);
19:
20:   return 0;
21: }
```

図 **A·1**　x^n の正しいプログラム

Fig. A·1　Correct program of x^n.

```
 1: #include <stdio.h>
 2:
 3: int main(void)
 4: {
 5:   double pow, x;
 6:   int n, i;
 7:
 8:   printf("x? ");
 9:   scanf("%d", &x);
10:   printf("n? ");
11:   scanf("%d", &n);
12:
13:   pow = 0;
14:   for (i = 0; i < n; i++) {
15:     pow = pow * n;
16:   }
17:
18:   printf("%f\n", pow);
19:
20:   return 0;
21: }
```

図 **A·2**　x^n の誤りプログラム

Fig. A·2　Error program of x^n.

を出力するプログラム．

- 誤り
  - 文字列の終わりまでの繰返しの条件 `*s != '\0'` を `*s == '\0'` にする．
  - if の条件式 `E1 && E2` を `E1 || E2` にする．
  - `else if` の `else` を削除する．
  - 文字列を入力する `scanf` 関数の第 2 引数の前に `&` がある．
- ダミーの編集可能箇所
  - `'\0'`
  - `printf` 関数の第 2 引数の前

A.2　誤り修正課題の例

図 **A·5** は文字列 s を逆順に表示する誤りのあるプログラムである．正しくは 7 行目の再帰呼び出しの実引数は s+1 であり，6 行目の `putchar(*s);` は 8 行目に記述しないとならない．問題が簡単になりすぎないように，文の削除に準じて，3 行目と 10 行目の空行をダミー編集可能箇所

```
 1: #include <stdio.h>
 2:
 3: void strconv(char *s)
 4: {
 5:   while (*s != '\0') {
 6:     if ('a' <= *s && *s <= 'z') {
 7:       *s = *s - 'a' + 'A';
 8:     }
 9:     else if ('A' <= *s && *s <= 'Z') {
10:       *s = *s - 'A' + 'a';
11:     }
12:     s++;
13:   }
14: }
15:
16: int main(void)
17: {
18:   char str[128];
19:
20:   printf("string? ");
21:   scanf("%s", str);
22:
23:   printf("%s => ", str);
24:   strconv(str);
25:   printf("%s\n", str);
26:
27:   return 0;
28: }
29:
```

図 **A·3**　文字列変換の正しいプログラム

Fig. A·3　Correct program of converting string.

```
 1: #include <stdio.h>
 2:
 3: void strconv(char *s)
 4: {
 5:   while (*s == '\0') {
 6:     if ('a' <= *s || *s <= 'z') {
 7:       *s = *s - 'a' + 'A';
 8:     }
 9:   _ if ('A' <= *s || *s <= 'Z') {
10:       *s = *s - 'A' + 'a';
11:     }
12:     s++;
13:   }
14: }
15:
16: int main(void)
17: {
18:   char str[128];
19:
20:   printf("string? ");
21:   scanf("%s", &str);
22:
23:   printf("%s => ", _str);
24:   strconv(str);
25:   printf("%s\n", _str);
26:
27:   return 0;
28: }
```

図 **A·4**　文字列変換の誤りプログラム

Fig. A·4　Error program of converting string.

```
 1: void putstrrev(char *s)
 2: {
 3:    ____________
 4:    if (*s == '\0') return;
 5:    else {
 6:       putchar(*s);
 7:       putstrrev(s-1);
 8:       ____________
 9:    }
10:    ____________
11: }
```

図 **A·5**　文字列 s を逆順で表示する誤りプログラム

Fig. A·5　Error program of putting string s in the reverse order.

© 2017 Information Processing Society of Japan

としている．

蜂巣 吉成　（正会員）

1994 年名古屋大学工学部情報工学科卒業．1999 年同大学院工学研究科情報工学専攻博士後期課程修了．同年南山大学経営学部情報管理学科助手，2000 年同大学数理情報学部講師を経て，現在，同大学理工学部准教授．この間 2004 年〜2006 年までエジンバラ大学客員研究員．博士（工学）．構造化文書の記述方法，ソフトウェアの開発支援環境，e ラーニングに関する研究に従事．日本ソフトウェア科学会，電子情報通信学会，日本教育工学会，IEEE Computer Society，ACM 各会員．

吉田 敦　（正会員）

1991 年名古屋大学工学部情報工学科卒業．1996 年同大学院工学研究科博士後期課程単位取得退学．同年豊橋技術科学大学知識情報学系助手，2000 年和歌山大学システム情報学センター講師，2009 年南山大学情報理工学部准教授を経て，現在，同大学理工学部教授．この間 2013 年〜2014 年までクィーンズ大学客員研究員．博士（工学）．プログラム解析および開発支援環境に関する研究に従事．日本ソフトウェア科学会，電子情報通信学会，IEEE Computer Society 各会員．

阿草 清滋　（正会員）

1970 年京都大学工学部電気第二学科卒業．同大学院進学．1974 年京都大学情報工学科助手．同講師，助教授を経て，1989 年〜2012 年まで名古屋大学教授．2014 年より南山大学理工学部教授．現在，同大学院理工学研究科長．（財）京都高度研究所所長を兼務．工学博士．ソフトウェア開発方法論，知的開発環境，仕様化技法，再利用技法，マンマシンインタフェース等の研究に従事．

© 2017 Information Processing Society of Japan