プログラミング学習における誤り訂正問題の自動生成の拡張に関する研究

2010SE030 古田優也　2010SE118 松下知将　2010SE155 丹羽紀裕

指導教員：蜂巣吉成

1　はじめに

プログラミング学習において，プログラムを記述すること（コーディング）や他人が書いたプログラムを読むこと（コードリーディング），プログラム中のバグを発見し訂正すること（デバッギング）が重要である．本研究ではデバッギング能力の向上について考える．

全文記述問題では，プログラム作成中にデバッギングを行なうが，プログラム中に含まれるバグの種類や数は学習者により異なり教員が体験してほしいと考えるデバッギング作業をしているとは限らない．空欄補充問題では，プログラムの大枠が完成していることから解答者はプログラムの流れを読み取る必要があり，コードリーディング能力の向上に向いている．しかし空欄補充問題には誤りが含まれておらずデバッギングを行えない．誤り訂正問題では意図的にバグを混入させることができるので，出題者が体験させたいと考えるデバッギング作業を行え，デバッギング能力の向上に向いている．

われわれの研究室では，プログラミング初学習者を対象に誤り訂正問題の生成，正誤判定の自動化の方法を提案している [2][4]．プログラミング初学習者の典型的な誤りを取り上げ，それらの誤りを混入させるための編集操作の観点から挿入，削除，置換，移動の 4 種類に分類している．正誤判定に関して，解答者の編集可能な箇所を制限することにより，解答パターンを限定し，複数解答にも対応している．これらの研究での問題点が 2 つある．

1. 取り上げている誤りが少ない
2. 編集操作で 4 種類あったが，削除と置換の誤りのみを対象としている

本研究ではこれらの問題を解決し，より実践的なデバッギング能力の向上のために，誤り訂正問題の生成方法と正誤判定の方法を拡張することを目的とする．問題点 1 を解決するために，教科書や講義資料から学ぶべき項目を列挙し，間違いやすい記述を誤りとして形式化した．結果，移動と挿入によって混入できる誤りが確認されたので，編集操作の移動と挿入を追加してそれらをパターンとして記述する方法と正誤判定方法を提案する．

提案する誤り訂正問題は次のように利用されることを想定している．全文記述問題や空欄補充問題と併用し，誤り訂正問題は，自習用課題として利用される．解答においては，学習者はコンパイルや実行などを行わずに，インスペクションにより誤りを発見し，訂正する．

正誤判定は正しいプログラムから抽出した正解と学習者の解答を比較することで行う．複数解答については，学習者が訂正したプログラムをコンパイル・実行し，出力結果から正誤判定を行う方法も考えられる．しかし，この方法では必要十分なテストケースを用意するための出題者の負担が大きい．誤り訂正問題では典型的な誤りのパターンを学習させたいという意図があり，その意図に合うように学習者を誘導したいが，テスト実行のみによる判定では訂正された記述を評価しないので，意図にあった学習をしているかが確認できない．以上のことから本研究では，実行結果の比較による正誤判定は行わない．

解答においては，コンパイラやデバッガは使用せずに，インスペクションにより誤りを発見し，訂正する．解答ときにデバッガを利用させて，デバッガの使い方を習得させることも考えられるが，本研究では，プログラムのロジックの把握とコードリーディング能力の向上を重視して，インスペクションのみでのデバッグを想定する．

2　関連研究

誤りを故意に混入させて，テストケースがそれを検出できるか評価する方法としてミューテーション法 [1] がある．この方法はテストケースが十分かどうかを測定することを目的としているので，学習意図に合わない誤りやプログラムの可読性を低くする誤り，コーディングでは起こりにくい誤りを混入させることがあり，誤り訂正問題には適さない．

アルゴリズム学習において誤り訂正問題の自動生成の方法を提案する研究 [3] が挙げられる．この研究ではアルゴリズムの理解度の向上を目的としている．複数のアルゴリズムから 5 つのパラダイムを抽出し，それぞれのパラダイム毎に誤りを定義する．出題者がソースコード，パラダイム，挿入する誤りの個数，生成する問題の数を選択することにより，誤り訂正問題の自動生成を行なっている．しかし，誤り訂正問題に対しての自動正誤判定方法は挙げられておらず，また問題がアルゴリズムのみであり，初学習者向けの文法の学習は考慮されていない．

3　誤り訂正問題の分類

誤りを列挙するにあたって，C 言語の教科書の単元毎によくある誤りをわれわれの経験と教科書 [5][6] の間違いやすい例や重要項目を基に抽出した．それらの誤りについて [2][4] の方法に基づいて編集操作と正解の種類の観点から分類し，編集可能箇所と複数解答の正誤判定方法を考える．

3.1　誤りの分類と学習意図

単元別に誤りを分類したところ基礎 2 個，条件分岐 4 個，繰り返し 6 個，配列 2 個，ポインタ 2 個，関数 4 個，文字列 3 個，構造体 3 個となった. 分類した一部を表 1 に示す. 誤りを混入させたプログラム例を図 1 に表す．左が模範解

表 1 誤りの分類

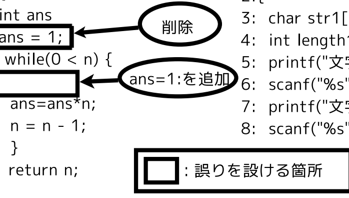
単元	項目	学習意図	誤り	編集操作	訂正可能箇所	正解数
文字列	文字列の入力	scanf 関数の文字列の入力	文字列の変数指定子に％ c を使用	置換	1 箇所	1 つ
文字列	文字列の入力	scanf 関数の文字列の入力	文字列でアンパサンドを使用	挿入	1 箇所	1 つ
文字列	文字列の構成	文字列の終端文字	終端文字の書き忘れ	削除	1 箇所	1 つ
繰り返し	初期化	繰り返し本体で累積して計算される変数の初期化文	初期化文の位置の間違い	移動	複数	1 つ
条件分岐	条件分岐の式	関係演算子	関係演算子の間違い	置換	1 箇所	複数
構造体	構造体メンバ参照	ポインタ型の構造体	ドット演算子とアロー演算子の間違い	置換	1 箇所	1 つ

```
 1: #include<stdio.h>
 2: int main(void)
 3: {
 4: char str1[128],str2[128],str3[128];
 5: int length1,length2;
 6:
 7: printf("文字列1を入力してしてください");
 8: scanf( "%s" , str1 );
 9: printf("文字列2を入力してください") ;
10: scanf( "%s" , str2 );
11:
12: for( length1 = 0;
13:         str1[length1] != '\0';lenth1++){
14:     str3[length1]=str1[length1];
15: }
16:
17: for( length2 =0 ; str2[length2] != '\0';
18:         length2++, length1++){
19:    str3[length1] = str2[length2];
20:  }
21: str3[length1] = '\0';
22: printf("%s \n" , str3);
23:
24: return 0;
25: }
```

```
 1: #include<stdio.h>
 2: int main(void)
 3: {
 4: char str1[128],str2[128],str3[128];
 5: int length1,length2;
 6:
 7: printf("文字列1を入力してしてください");
 8: scanf( "%c " ,& str1 );
 9: printf(" 文字列2を入力してください") ;
10: scanf( "%c "  ,& str2 );
11:
12: for( length1 = 0;
13:         str1[length1] == '\0'; lenth1++){
14:     str3[length1]=str1[length1];
15: }
16:
17: for( length2 =0 ; str2[length2] == '\0';
18:         length2++, length1++){
19:    str3[length1] = str2[length2];
20:  }
21: 
22: printf(" %c \n" , str3);
23:
24: return 0;
25 }
```

図 1 2 つの文字列を入力し，1 つの文字列に結合して出力するプログラム

```
 1:int fact(int n)
 2: {
 3: int ans
 4: ans = 1;
 5:  while(0 < n) {
 6: 
 7:    ans=ans*n;
 8:    n = n - 1;
 9:   }
10:   return n;
11:}
```




```
1:int main(void)
2:{
3:  char str1[128],str2[128],str3[128]
4:  int length1,length2;
5:  printf("文字列1を入力してください");
6:  scanf("%s" & str1);
7:  printf("文字列2を入力してください");
8:  scanf("%s" & str2);
```

図 2 左:移動操作の例，右:挿入操作の例

答，右が誤りを含んだプログラムである．四角の中に誤りを含ませた．

3.2 編集操作の分類

編集操作の分類を [2][4] と同様に削除，置換，移動，挿入の 4 つに分類した結果，削除 11 個，置換 13 個，移動 2 個，挿入 1 個となった．問題を追加したことにより，既存の研究にはない移動と挿入を本研究では扱う．図 2 の左側は，ある 1 行を別の行に移動させる例で，図 2 の右側は特定の箇所に字句を追加する例である．

3.3 正解の分類

編集操作の削除・置換・挿入についての正解の分類を行ったところ，[2][4] と同様に (a) から (d) に正解の分類ができた．また，編集操作の移動に対しては新たに正解の分類 (e) を作る．

(a) 訂正箇所は 1 つで，正解が 1 つの誤り
(b) 訂正箇所は 1 つで，正解の数が複数の誤り
(c) 訂正可能箇所は複数で，正解が 1 つの誤り
(d) 訂正可能箇所は複数で，正解が複数の誤り
(e) 文を削除し，訂正箇所が複数で正解が複数ある誤り

(a) は誤りを混入させた箇所を正解と同じ記述に訂正することで正解となる．例えば文字列の %s を %c に置換する問題では，誤りの部分を元の字句に訂正することで正解となる．

(b) は正解となる記述箇所は 1 つしかないが正解の数が複数あるときである．例えばで a+b と b+a は記述の仕方は違うが計算結果は同じになり，元の字句とは違う記述でも正解となる．また相関性のある問題もある．例えばループの初期化の置換で for(i=0;i<5;i++) を for(i=1;i<5;i++) と置換の誤りを混入させたとき，for(i=1;i<=5;i++) など，元の字句と異なる可能性が生まれ，正解が複数となる

(c) は正解の記述は 1 つしかないが，それを記述しても正解となる記述箇所が複数あるときである．例えば累乗を計算するプログラムで，繰り返し本体で累積して計算される変数の初期化文を削除したときに生じる．この場合は削除された一文から前に初期化文を追加することで正解となる．

(d) は編集可能箇所を多く設定したときに，誤りを混入させた箇所とは異なる箇所を訂正しても正解となる場合である．出題するプログラムに対して編集可能箇所が多くなると，出題者の意図しない別解が出じる場合や他の編集可能箇所を訂正することで他の誤りに対しても影響がある相関性の誤りがでる．

(e) は文を削除して別の行にその文を移動したときに生じる．誤った位置に移動した文を削除して，正解となる行に文を追加することで正解となる．例えば，繰り返し本体で累積して計算される変数の初期化文を繰り返し内に移動したときに生じる．この場合は移動した初期化文を削除して繰り返しより前に初期化文を追加することで正解となる．

3.4 編集可能箇所

編集可能箇所の数が少ないと誤り箇所が特定しやすくなり，多くなると出題者の意図しない別解が生じる場合がある．よって，正解の分類 (a) から (e) に編集可能箇所を系統的に決めることで，解答の推測をできないようにする．系統的に決められない種類の誤りについては，例外の編集可能箇所を決める．(d) については，別解を自動で正誤判定するのは難しいので，出題者が確認して編集可能箇所を調整し，別解が生じないようにする．

(a) は別解がなく元の字句を記述することで正解となる．

編集可能箇所を探すことで誤り箇所がわかる可能性があるので，類似した箇所を編集可能箇所とする．例えばポインタ変数の間接演算子の削除の場合，同じ変数の直前を編集可能箇所にすることで誤りの箇所を推測できないようにする．

(b) は元の字句とは別の記述をしても正解となるが，正解となる記述箇所が 1 つのときである．すでに別解があるので編集可能箇所は他に設定をしない．for 文が複数あるとき，一方の for 文では初期化の部分が編集可能箇所で，他方では制御構文が編集可能箇所となっているような一貫した編集可能箇所でないとき，学習者は編集可能箇所から誤りが特定できるので，for 文の場合であれば 3 つの式の同一部分に誤りを混入させ，編集可能箇所として統一し，編集可能箇所から誤り箇所が推測できないようにする．

(c) は元の位置から前の空行の編集可能箇所に正解の記述をすることで正解となる．正解となる箇所のみが編集可能箇所では容易に答えが特定されるので，制約条件として処理毎に空行があることを前提とし，空行を編集可能箇所と設定し，答えを推測できないようにする．

(e) は移動された文を削除し，削除した文を元の位置から前の空行の編集可能箇所に正解の記述をすることで正解となるので，移動された文の全体を編集可能箇所として設定を行い，移動される前の箇所も編集可能箇所として設定を行う．さらに (c) と同様に答えが推測できないように処理毎の空行を編集可能箇所とする．

3.5　正誤判定方法

3.1 節で挙げた誤りについての正誤判定方法を本節で述べる．次の (a)，(b)，(c)，(e) は 3.2 節で挙げた分類に対応している．

(a)，(b) は正解となる記述箇所は一箇所であるので，同じ箇所に正解となる記述を書いたとき正解とする．(b) のとき正解となる記述が複数通りあるので，正誤判定用 CGI に，解答パターンを記述することで複数解答に対応する．

(c) は元の位置から前の空行の編集可能箇所に正解の記述をすることで正解とする．模範プログラムに「プログラムの中で初期化を必要とする変数の位置を，プログラムが正しく動作する範囲で，可能な限りその変数を用いた命令文の直前に行う」という制約条件を設けることで，誤りを混入した場合，同一箇所，またはそこよりも前の位置に記述することで正解となる．

(e) は移動された文を削除し，削除した文を (c) と同様に元の位置から前の空行の編集可能箇所に正解の記述をすることで正解とする．

4　誤り訂正問題自動生成システム

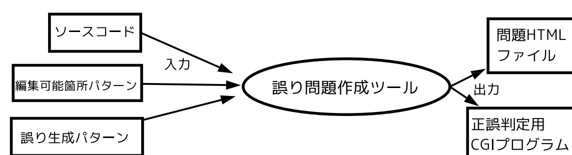


図 3　誤り訂正問題の生成方法

本研究では字句の書き換えツールである TEBA[7] を用いて誤り訂正問題の生成を行い，それらの解答には CGI を用いて正誤判定を行なう．模範解答を作成し問題生成ツールを用いることで問題の生成を行う．模範解答に適用する誤りパターンと編集可能箇所パターンは出題者がツールを使用するときに選択し，3.4 節であげた編集可能箇所と誤りを用いて訂正問題が作成される．編集可能箇所にもパターンを使用する．パターンを用いることにより汎用性が増し，様々なプログラムに対して適用可能となる．図 3 はシステムの一連の流れである．

誤りを含む問題の出題と解答には，既存研究と同様に HTML 形式で Web を用いる．HTML で編集可能箇所にそれぞれ ID がつけられ，解答者の編集操作後に HTML から編集可能箇所の抽出を行ない，CGI を用いて模範解答と ID，訂正字句と正解を比較して正誤判定を行なう．

4.1　誤りパターンの記述

本研究では誤りを混入させるにあたって，どこに，どのように混入させるかパターン化する．図 4 は，誤り訂正問題の作成に使用する誤りパターン例である．図 4 の左のパターンについて，%before から %after（以下 before 部）が正しいプログラムで，%after から %end（以下 after 部）が誤りを含むプログラムである．編集可能とする字句を’<@’ と’@>’ でマークアップし，誤りはその位置に混入され after 部のようにプログラムが書き換わる．before 部でマークアップされた字句を正解として正誤判定用 CGI に書き込み，同様の解答にすることで正解となる．編集可能箇所パターンは誤りパターンと同様に，編集可能とする字句を’<@’ と’@>’ でマークアップする．before 部と after 部を同じ記述にすることにより編集可能箇所パターンを作成する．

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

```
% before
<#@ ${id1:ID_FVAR} = 0; @>
for(${e:EXPR}){
  <#@   @>
  ${s:STMT*}$ ;
}
% after
<#@   @>
for(${e}){
  <#@ ${id1} = 0; @>
  ${s}$ ;
}
% end
```

図 4　パターンの記述方法

4.2　挿入の誤り

図 4 の左のパターンが挿入のパターン例である．before 部では追加する空白の箇所を’<@’ と’@>’ でマークアップし，after 部の’<@’ と’@>’ 内に追加したい文字を記述することで，パターンを作成する．

4.3　移動の誤り

既存研究 [2][4] では誤りの混入位置を’<@’ と’@>’ でパターン記述をしていたが，移動の誤りについては#を含ん

だ'<#@' と'@>' で記述する．図 4 の右のパターンが移動のパターン例である．移動の誤りとは 3.5 節で挙げたように一文の削除と記述を同時に行なうことにより正解となる．#の記述によりある特定の誤りの間に関連性をもたせることができる．移動の誤りを他の誤りと異なる ID をつけることで正誤判定用 CGI 内部でも他との区別が可能になり，正誤判定が可能となった．移動の誤りは一つのプログラムの中に一つまでとする．これについては 5 章で考察する．

5　考察

本研究では，新たな編集操作として移動と挿入を追加した．複数の編集操作を行なった際の編集可能箇所について考察する．図 5 は複数行連続して移動や削除をする場合である．図のように編集可能箇所が連続していることによりプログラムが可読性が低くなり，誤り箇所を見つけることが容易になってしまう．このようなプログラム例が生成されるのは，削除と削除，移動と移動，削除と移動の編集操作を連続する複数行に対して行なったときである．以下で，図 5 のような場合の対応方法を考察する．

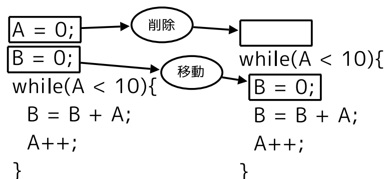


図 5　移動と削除が 2 行連なる場合

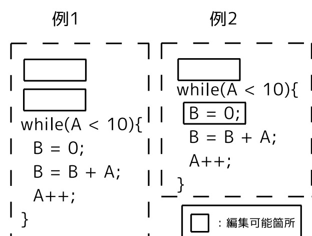


図 6　編集可能箇所の設定例

図 6 例 1 は，編集可能箇所を編集操作の削除と移動を行なった 2 行とも設定する場合である．制約条件として，セミコロンと代入演算子は 1 行に一つまでとする．この場合，2 行連続して編集可能箇所となるので解答者に誤りがあると推測されやすくなり，訂正する箇所を見つけやすくなる．

図 6 例 2 は，編集操作の削除と移動を 2 行分行なったが，編集可能箇所は 1 行のみとする場合である．この場合，2 通り考えられる．ひとつは，制約条件としてセミコロンは一つで，操作した行の右辺が等しい場合である．例 1 のように代入演算子の制約条件は設けないので，解答の例は A = B = 0; などとなる．誤りの位置は特定することは難しいが，右辺が等しいという制約条件により，問題の汎用性が低下する．もうひとつは，制約条件は設けない場合である．正解の例としては，先の解答に追加して，A = 0; B = 0; といった 2 文を 1 行に記述することで正解となる．正解の数の増加により解答の特定は難しくなるが，出題者が正誤判定ツールの正解パターンを全て記述する必要があることから出題者の負担が多くなる．本研究では問題生成の汎用性を保ちつつ，正誤判定が可能な範囲を検討した結果，例 1 が適切だと考えた．

他にも解答者の解答に応じて動的に編集可能箇所を増やしていく方法が考えられる．これにより，誤りの位置を特定することが難しくなるが，解答の自由度が増し，予期せぬ解答が出てくることが想定される．加えて，増えた編集可能箇所に対しての正誤判定方法などシステム全体の検討が必要になる．

6　おわりに

本研究では教科書を基に単元毎に誤りを列挙し，[2][4] では対応していない誤りを追加した．追加した誤りから編集操作についての分類を行い，挿入と移動を発見した．移動と挿入について誤りを混入させるためのパターン記述や正誤判定方法，編集可能箇所の設定についての提案を行った．

今後の課題として，教科書 [5][6] 等の基礎的な文法の誤りだけではなく，アルゴリズム等のプログラムを対象とした誤り訂正問題の出題と，実際に問題を解いてもらい，編集可能箇所の数や間違いやすい誤りであるかどうか，解答に対して正しく正誤判定が行なわれたかどうかを検証することが挙げられる．

参考文献

[1] Jia，Y．and Harman，M．：An Analysis and Survey of the Development of Mutation Testing，IEEE Transactions on Software Engineering, Vol．37, No. 5(2011)，pp.649-678.

[2] 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)，(2013)，pp.35-40.

[3] 長瀧寛之，伊藤亮太，et al，“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”，情処学論，Vol．49，No．10(2008)，pp.3366-3376.

[4] 小川和輝，佐藤雄基，“プログラミング学習における誤り訂正問題の自動生成方法に関する研究”，南山大学情報理工学部ソフトウェア工学科，2012 年度卒業論文要旨集.

[5] 柴田望洋，「新版明解 C 言語　入門編」，ソフトバンククリエイティブ株式会社，2012.

[6] 清水忠昭，菅田一博，「新-C 言語のススメ C で始めるプログラミング」'，株式会社サイエンス社，2010.

[7] 吉田 敦，蜂巣 吉成，et al，“属性付き字句系列に基づくプログラム書き換え支援環境”，情処学論，Vol．53，No.7，pp.1832-1849，2012.