C 言語プログラム学習者の誤り指摘ツールの提案
ー記述適正率に基づいた指摘メッセージの生成ー

2016SE020 石橋友哉　2016SE053 松本和也　2016SE073 砂川涼平
指導教員：蜂巣吉成

1　はじめに

C 言語プログラムを学習している中で，学習者は行き詰まったり，モチベーションが低下する場合がある．その原因として自身が作成したプログラムのコンパイラのエラーメッセージや警告の内容を読み取ることができなかったり，エラーメッセージや警告が出ない誤りをしたことで，誤りの場所や修正方法がわからないということが挙げられる．

これらのことを解決するために，プログラムの誤りの場所を特定したり，修正方法を提案するツールや先行研究がいくつか提案されている [1][2]．これらは，使用者全員に対して誤りを説明した指摘を行っていたり，修正方法を具体的に示した指摘を行っており，指摘内容を理解できない者や誤りの修正方法を理解せずにソースコードを機械的に修正する者が出てくるという問題点がある．

本研究では，支援を行う誤り毎にこれまでのコーディング履歴から記述適正率を測り，その値に応じた指摘を行う方法を提案する．記述適正率とは，学習者が起こしやすい誤り毎に，過去に記述できていた割合をコーディング履歴を元に算出した値である．記述適正率と事前に決められたしきい値を用いて誤り毎に学習者を分類し，学習者に応じた指摘メッセージを出力する．記述適正率については，4 章で詳細に述べる．

これまでに，正しい記述があまり行えていない学習者の記述適正率は低くなる．このような学習者は，コンパイラのエラーメッセージや警告を見ても，専門用語が含まれており修正が行えないと考えられるので，修正を促すような具体的な指摘を行う．正しい記述を多く行えている学習者は記述適正率が高くなる．このような学習者は，誤りがあった場合でもケアレスミスと考えられるので，誤りを見つけられるような具体的な指摘を行う．正しい記述も行えているが誤った記述をした場合に，自ら修正を行える問題と行えない問題が混在しているような学習者の記述適正率は低と高の間になる．このような学習者は，まだコンパイラのエラーメッセージや警告を見た際に，理解できる指摘とできない指摘があると考えられる．一般的に，記述適正率が高い学習者から記述適正率が低い学習者になるほど具体的な指摘を行なうことが考えられるが，本研究では記述適正率が中の学習者へは，機械的に修正することを防ぎ，正しい記述方法の定着を促すために自身がした誤りについて考えられるような抽象的な指摘を行う．

今回提案するツールで検出を行う誤りの中には，問題に依存する誤りがある．本研究で提案するツールで検出を行える誤りのみでは，網羅できていない．そこで，パターン記述を用いて，検出できる誤りを追加できる仕組みを実現した．また，パターン記述と課題番号を用いれば問題に依存するような誤りも検出できる．

本研究では，指摘する誤りを分類しそれに対する指摘メッセージを整理した．記述適正率を定め，パターンマッチングを用いて誤りを指摘するツールの設計・実現をした．分類した誤りをパターン記述で容易に追加・検出できる仕組みを実現した．

2　関連研究

大須賀ら [1] は，C 言語プログラムに対してコーディング規約を満たしているかをチェックするためのコーディングチェッカとして CX-Checker を提案している．CX-Checker は，XPath を用いたルール，ルール記述用 API を用いたルール，DOM を用いたルールの 3 つのルールでコーディング規約を自由にカスタマイズすることができる．

初学者を対象としたフィードバックを行うツールとして，内田ら [2] は C-Helper を提案している．C-Helper は，C 言語のソースコードを静的に解析し，ソースコードの問題点や解決策を初学者にとって分かりやすいメッセージとして出力することで，プログラミング学習を支援している．

C-Helper では，以下の問題などを検出できる．

- printf() のパラメタ不整合
- return 文の不足

メッセージ例は次のようになる．

- 引数と%変数の型が合いません．(浮動小数点数の表示には%f，%e，%g などが使えます．)
- return 文がありません．

上述の関連研究では，学習者に指摘を行う際にプログラムの誤り毎に単一のメッセージしか出力しない．本研究で提案するツールは，学習者を低・中・高に分類し，それぞれに異なるメッセージを出力する．これにより，指摘内容を理解できない者が減少したり，機械的にプログラムの誤りを修正することを防ぐことができる．

3　指摘するソースコードの誤りとメッセージ

3.1　ソースコードの誤り例

コンパイラのエラーメッセージや警告に専門用語が含まれていて，学習者にとって修正が困難であるもの，エラーメッセージや警告は表示されないが，プログラムが意図通りの結果を返さないもの等の学習者によくある誤りについて支援を行う．学習者によくある誤りで支援を行う誤りは，12 項目ある．その中から 4 つの例を以下に挙げる．

項目 1 if 文の制御式が “a<b && b<c” ではなく “a<b<c” と記述している．

項目 2 “s++” と記述すれば良い所で，“*s++” と記述している．

項目 3 除法を行う際に，“double 型=int 型/int 型” という式で計算をしている．

項目 4 “a/(b*c)” と記述すべき所で，“a/b*c” と記述している．

項目 2，項目 3，項目 4 について具体的な誤りの例を以下に示す．

ソースコード 1 文字列が等しいかを判定するプログラムでの例 (項目 2)

```
1 int str_equal(char *s, char *l){
2     while(*s != '\0'){
3         if(*s != *l)
4             return 0;
5         *s++;
6         *l++;
7     }
8     return *s == *l;
9 }
```

ソースコード 2 配列の平均を求めるプログラムでの例 (項目 3)

```
1 int array_ave(int array[], int num){
2     int i, sum = 0;
3     double ave = 0;
4     for(i = 0;i < num;i++){
5         sum += array[i];
6     }
7     ave = sum / num;
8     return ave;
9 }
```

ソースコード 3 BMI を求めるプログラムでの例 (項目 4)

```
1 double bmi(int height, int weight){
2    double bmi;
3    bmi = weight / height * height;
4    return bmi;
5 }
```

ソースコード 1 は，参照外しを行わなくても良い 5 行目，6 行目で参照外しを行っている例である．

ソースコード 2 は，7 行目で整数型である “sum” と “num” で計算を行った結果を実数型の “ave” に代入しているので，小数点以下が切り捨てられた結果が返される例である．

ソースコード 3 は，3 行目で BMI の計算を行なう際に “体重/(身長*身長)” という式ではなく “体重/身長*身長” という式で計算を行っているので，期待した計算結果が返されない例である．

3.2 記述適正率に応じたメッセージ

3.1 章で挙げたプログラムの誤りに対して，3.1 章の項目毎にメッセージによる指摘を行う．指摘を行う際に，4 章の (1) 式から得られる値としきい値を用いて，学習者を低・中・高に分類しそれぞれに適したメッセージを出力する．

これまでに正しい記述をあまり行えていない学習者の記述適正率は，低くなる．このような学習者は，プログラムの誤りに対してコンパイラのエラーメッセージや警告を見ても，専門用語が含まれており修正が行えない．そこで，誤りの種類，誤りの場所，どのような誤りであるかや誤りの修正案を提示することで，誤りの修正ができずに行き詰まることを防ぐ．

これまでに正しい記述を多く行えていた学習者の記述適正率は，高くなる．このような学習者は，誤りがあった場合でもケアレスミスであると考えられる．よって，誤りの種類，誤りがある場所を提示することで修正を促す．

これまでに正しい記述を行なうこともできている．しかし，誤った記述をした際に，自ら誤りを特定し修正を行える問題と行えない問題が混在しているような学習者の記述適正率は，低と高の間になる（記述適正率が中とする）．このような学習者は，まだコンパイラのエラーメッセージや警告を見た際に，理解できる指摘とできない指摘があると考えられる．また，正しい記述を定着しきれていないということも考えられる．一般的に，記述適正率が高い学習者から記述適正率が低い学習者になるほど具体的な指摘を行なうことが考えられるが，本研究では記述適正率が中の学習者へは，誤りの種類，その誤りをプログラム内で何回誤っているかを提示する．これは，コンパイラのエラーメッセージや警告を見て，自身が作成したプログラムの誤りに対して自ら考えて修正することを促すことで，ただ指摘を見て機械的に修正することを防ぎ，コンパイラのエラーメッセージや警告の意味を理解した上で正しい記述方法を定着させることができると考えたからである．

このように指摘内容を変更することで，学習者の学習段階に合った指摘を行える．実際の出力メッセージの例を以下に示す．

低：<“a/(b*c)” と記述すべき所で，“a/b*c” と記述している間違いについて>
3 行目が誤っています．
今回の問題では “a/b*c” のような式は用いません．
値がずれる場合があります．
“ / ” と “ * ” の優先順位が同じのため，左から計算されます．
(例)： 6 / 2 * 3 = 9
6 / (2 * 3) = 1
中：<“a/(b*c)” と記述すべき所で，“a/b*c” と記述している>
1 箇所誤っています．
高：<“a/(b*c)” と記述すべき所で，“a/b*c” と記述している>
3 行目が誤っています．

図 1 項目 4 の指摘メッセージ例

4 記述適正率について

学習者に適した支援をするにあたり，3.1 章の項目毎に学習者がどの程度理解しているかを数値化して知る必要がある．しかし，学習者がどの程度理解しているかを数値として知ることは困難である．そこで，学習者に適した支援を行うために 3.1 章の項目毎に記述適正率を定める．

本研究では，どの程度プログラムを正しく記述できているかを記述適正率と定め，3.1 章の項目毎に，過去の正誤判定の結果を用いて (1) のような式で求める．また，記述適正率の算出には加重平均の考え方を用いる．本研究では，直近のデータの重みを大きくし，過去のデータの重みを小さくする．

この値を元に，項目毎に学習者を分類し図 1 のようなメッセージを出力する．

$$x_L(n_L) = \begin{cases} \frac{\sum_{i=1}^{n_L} w_i a_L(n_L - i + 1)}{\sum_{i=1}^{n_L} w_i} & (n_L \leq d_L) \\ \frac{\sum_{i=1}^{d_L} w_i a_L(n_L - i + 1)}{\sum_{i=1}^{d_L} w_i} & (n_L > d_L) \end{cases} \quad (1)$$

- L： 3.1 章の項目
- n_L： ツールを使い始めてからの 3.1 章の項目 L が含まれているコンパイル回数（$n_L \geq 1$）
- $x_L(n_L)$： 3.1 章の項目 L の n_L 回目のコンパイル時の記述適正率.（$0 \leq x_L(n_L) \leq 1$ ）
- $a_L(i) = \{0, 1\}$： 3.1 章の項目 L の i 回目の判定箇所の正誤.ただし，正しい判定は 1，誤った判定は 0 とする.
- d_L： 3.1 章の項目 L の計算に使用する最大のデータ数.
- w_i： i 番目の重み.（$0 < w_i < 1$ ）

3.1 章の項目 1 において，正誤判定履歴が過去の判定から順に「X，O，O」（O は正の判定，X は誤りの判定）であり，重みを「1，0.8，0.4，0.2」とする場合，式 (1) でどのように記述適正率が求められるかを述べる．

この場合，$n_1 = 3$ となり，$a_1(1) = 0$, $a_1(2) = 1$, $a_1(3) = 1$ となる．また，重みは $w_1 = 1$, $w_2 = 0.8$, $w_3 = 0.4$, $w_4 = 0.2$ となる．この時の記述適正率は，

$$\frac{a_1(3) * w_1 + a_1(2) * w_2 + a_1(1) * w_3}{w_1 + w_2 + w_3} = 0.818 \quad (2)$$

となる．

5 誤り指摘ツールの提案

5.1 ツールの概略

本研究では，記述適正率に応じた誤り指摘ツールを提案する．学習者は，実習時や自習時に WebIDE を用いて解答を行う．その際に，学習者は WebIDE 上の画面で課題番号を選択し解答を行う．課題番号は，3.1 章の項目 4 のような誤りの判定を行う際に使用する．項目 4 はすべての課題で検出を行うと，“a/b*c” のような記述とマッチする場合に，学習者の意図通りであったとしてもすべて誤りの判定になる．よって，検出する課題を指定し，特定の場合のみで検出を行う必要がある．学習者が WebIDE を用いてコンパイルを行うタイミングで，学習者の解答と課題番号をツールに受け渡す．受け取った学習者の解答に対して 3.1 章で挙げた項目毎に正誤判定を行い，誤りの判定だった場合，誤りの判定があった項目の正誤判定履歴を用いて記述適正率を計算する．記述適正率の結果から低・中・高の三段階に分類し，項目毎に各段階に応じた指摘メッセージを出力する．

本研究で提案するツールでは，12 項目の誤りを指摘することができる．しかし，12 項目だけでは学習者が起こす誤りを網羅していないため，検出を行う項目の拡張を行える仕組みが必要である．項目を追加する際に，容易にパターン記述を行うことのできる TEBA[3][4] を使用する．TEBA を用いた場合，パターン記述では構文解析を用いて検出できる誤りのパターンを記述することができる．また，型の情報を用いてパターン記述をできるように TEBA の拡張を行った．よって，教員が新たに項目を追加する場合は，構文解析のみを用いる場合，構文解析と型の情報を用いて検出できる誤りのパターンを記述することで追加できる．

5.2 ツールの設計

本研究で提案するツールを設計するにあたって，内部の構造を図 2 のように設計を行った．ここで提案するツール内部の処理の流れについて述べる．ツールは，学習者の解答と課題番号を受け取り，受け取ったソースコードに対して正誤判定を行う．正誤判定は，予め教員が記述した項目毎のパターンを用いて行う．正誤判定の結果は，項目毎に正誤判定履歴として判定履歴データベースに保存される．また，正誤判定の結果が誤りだった項目は，指摘メッセージを出力する必要があるため，その項目の過去の正誤判定履歴を用いて記述適正率を計算する．その後，記述適正率としきい値を用いて学習者が低・中・高のどの段階に当たるかを確認し，その段階に応じたメッセージを出力する．

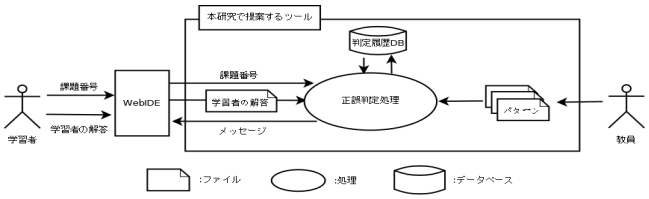

図 2 ツールの内部構造

5.3 パターンの記法

提案するツールでは，誤りの指摘を行うために予め指摘する誤り毎にパターンファイルを記述する必要がある．パターンファイルには「正誤判定パターン」,「記述適正率別のメッセージ」が記述されている．「正誤判定パターン」には，正しい記述とマッチするパターン，誤りの記述とマッチするパターンのそれぞれが記述されている．3.1 章の項目 4 では，正しい記述とマッチするパターンは “${:ID_VF}/(${:ID_VF}*${:ID_VF})” のように記述し，誤りの記述とマッチするパターンは “${:ID_VF}/${:ID_VF}*${:ID_VF}” と記述する．正しい記述とマッチするパターンは，学習者が過去に正しい記述を行っていたことを確認するために必要である．指摘メッセージは低・中・高のそれぞれ記述する必要がある．

図 1 の指摘メッセージのうち，高のメッセージを出力する場合は,「<“a/(b*c)” と記述すべき所で，“a/b*c” と記述している>${LINE} 行目が誤っています.」と記述を行う.「記述適正率別のメッセージ」の記述で，何行目でマッチしたかを表示する “${LINE}”，マッチした回数を表示する “${COUNT}” を用いることができる．また，1 章で述べた検出を行う誤りを追加する仕組みについては，先述のパターンファイルを作成し，項目を追加できる．

6 評価・考察

6.1 パターンの記述能力

本研究で提案するツールで使用するパターンは以下のことを記述することができる．

- パターン変数は，任意の識別子，型，文，宣言子，宣言に適応した変数で，${:型名 } の様に記述できる．
- グループとして記述の繰り返しや存在しない場合があるものを，正規表現で用いる “ *，?，+ ” を用いて表現できる．
- 変数の型の情報を使用したパターンを記述することができる．

現在のパターン記述では，3.1 章の項目 3 を検出するために，“${:ID_VF_DOUBLE}=${:ID_VF_INT}/${:ID_VF_INT}” のパターンでマッチさせる．このパターン記述では “$ans = num1/num2$” のような誤りを検出することは可能だが，“$ans = num/2$” のように数値とマッチさせることはできない．よって，式の型の情報をパターン記述できれば数値ともマッチすることが可能になる．

6.2 記述適正率の検証

4 章で定めた記述適正率は直近の正誤判定履歴を重視して算出される．

本研究で提案するツールでは，過去のソースコードの正誤判定を用いるため，長期間ツールを使用する必要がある．しかし，長期間の検証は困難であるため，今回の検証では架空の人物を設定して行った．また，記述適正率で用いる適当な重みとしきい値の設定を行い，架空の人物の段階の変化の検証を行った．使用する記述適正率の重みは「1，0.8，0.4，0.2」であり，しきい値は「0.56，0.70」架空の人物 A〜C の設定を以下に示す．

- A：コンパイラのエラーメセージや警告を読み取ることができる．演習問題まで解くことができる．
- B：コンパイラのエラーメッセージや警告を少し読み取ることができる．例題は容易に解くことができるが，演習のような問題で躓く場合がある．
- C：コンパイラのエラーメッセージや警告を読み取ることができない．例題で躓く場合がある．

各人物は，“a <b && b <c” を扱う内容が含まれている課題を 4 問解いたとする．検証の結果を表 1 に示す．表 1 の正誤判定履歴は左のデータほど新しく，正の判定結果の場合は O，誤りの判定結果の場合は X と表す．

表 1 より，正誤判定履歴で直近に正の判定が多い人物は記述適正率が高くなり，誤りの判定が多い人物は記述適正率が低くなることが確認できた．

人物 A は，課題 4 で誤った記述を行ったが課題 1〜3 までは正しい記述を行えていたため，高のメッセージが出力された．人物 B は課題 4 で誤った記述を行ったが，課題 2，3 で正しい記述をしていたため，高のメッセージが出力された．人物 C は，課題 1，2 では低のメッセージを見ながら修正を行った．課題 3 で，最初に誤った記述を行った際は，直近の重みが大きいため中のメッセージで指摘が行われた．次に誤った記述を行った際には，直近の正誤判定履歴に誤りの判定が続いたため低のメッセージが出力された．

今回の検証では，記述適正率にばらつきがあったが，長期間ツールを使用し検証を行った場合，記述適正率は次第に収束していくと考えられる．よって，実際に長期間ツールを使用し，記述適正率がどのように変化していくか確認する必要がある．

表 1 検証の結果

人物	正誤判定結果	正誤判定履歴	課題番号	記述適正率	段階
A	O		1	-	-
	O	O	2	-	-
	O	O O	3	-	-
	X	O O O	4	1.000	高
	O	X O O O	4	-	-
B	X		1	0.000	低
	O	X	1	-	-
	O	O X	2	-	-
	O	O O X	3	-	-
	X	O O O X	4	0.917	高
	O	X O O O	4	-	-
C	X		1	0.000	低
	O	X	1	-	-
	X	O X	2	0.556	低
	O	X O X	2	-	-
	X	O X O X	3	0.583	中
	X	X O X O	3	0.417	低
	O	X X O X	3	-	-
	X	O X X O	4	0.500	低
	O	X O X X	4	-	-

7 おわりに

本研究では，記述適正率に応じた誤り指摘ツールを提案した．学習者のコーディング履歴から指摘を行うソースコードの誤りに該当する箇所をパターンマッチングにより検出し，正しく記述できているか否かを判定することで記述適正率を算出した．その値としきい値から学習者を分類しそれぞれに応じたメッセージを出力することができた．

今後の課題として，記述パターンの拡張と，重み・しきい値を定めることが挙げられる．これらを定めるために長期間ツールを使用し，重み・しきい値の適切な値について検討する必要がある．

参考文献

[1] 大須賀 俊憲，小林 隆志，渥美 紀寿，間瀬 順一，山本 晋一郎，鈴村 延保，阿草 清滋：CX-Checker：柔軟なカスタマイズが可能な C 言語コーディングチェッカ, 情報処理学会論文誌，Vol.53，No.7，pp.590-600 (2012)

[2] 内田 公太，権藤 克彦：C 言語初学者向けツール C-Helper の現状と展望，第 54 回プログラミング・シンポジウム予稿集，Vol.2013，pp.153-160 (2013)

[3] 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)

[4] Yoshida，A. and Hachisu，Y.：A Pattern Search Method for Unpreprocessed C Programs based on Tokenized Syntax Trees，Proc. 14th IEEE International Working Conference on Source Code Analysis and Manipulation，SCAM2014，pp.295-304 (2014)