Vol.2014-CE-125 No.3
2014/6/7

プログラミング演習における コーディング状況把握方法の考察

蜂巣 吉成[1,a)]　吉田 敦[1,b)]　阿草 清滋[1,c)]

概要：情報系大学のプログラミング演習では，教員は学習者のコーディング状況を把握して，学習者に合った適切な指導を行うことが求められる．しかし，限りある演習時間内に学習者全員のコーディング状況を把握することは難しい．本研究では，演算子，式，文，文の構造といった観点から，学習者のソースコードと模範解答プログラムを比較し，分析を行うことでコーディング状況を把握する方法を提案する．学習者の編集途中のソースコードを随時取得するために Web を用いた開発環境を実現し，演習問題数題に適用したところ，コーディング状況の把握が可能であることを確認した．

キーワード：プログラミング演習，コーディング状況

Understanding Coding Processes in Programming Exercises

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

Abstract: In programming exercises at universities, lecturers can give learners good advice by understanding learners' coding processes. It is, however, difficult to understand all learners' processes during the limited exercise course hours. In this paper, we propose a way for understanding coding processes by comparing learners' codes and correct programs prepared by lecturers from the views of operators, expressions, statements, and structures of statements. We have implemented a Web integrated development environment to collect learners' editing codes. We have applied our approach to some exercises and confirmed its effectiveness.

Keywords: Programming Exercise, Coding Process

1. はじめに

情報系大学などのプログラミング教育では，学生 (以下，学習者と呼ぶ) にプログラムを記述させる演習形式が一般的である．演習時間において学習効果をあげるためには，教員が学習者のコーディング状況を把握してアドバイスを与えることが重要である．

例えば，C 言語で要素数 `size`（`>0`）の整数配列 `d` の最小値を求める関数 `minIntArray` を作成する演習課題におけるコーディング状況とアドバイスについて考える．次のプログラムを模範解答とする．

```
int minIntArray(int d[], int size)
{
   int min, i;
   min = d[0];
   for (i = 1; i < size; i++) {
      if (d[i] < min) {
         min = d[i];
      }
   }
   return min;
}
```

演習開始後，ある程度時間が経過しても for 文の中の if 文が記述できない学習者が多数いるならば，配列を走査しながらこれまでの最小値と配列の要素を比較する，といったヒントを学習者全体に与え，少数ならば個別に指導する．繰り返しの範囲を，`i <= size` と誤って記述している学習

[1] 南山大学理工学部ソフトウェア工学科
27 Seirei-cho, Seto-shi, Aichi-ken, 489–0863, Japan
[a)] hachisu@nanzan-u.ac.jp
[b)] atsu@nanzan-u.ac.jp
[c)] agusa@nanzan-u.ac.jp

ⓒ 2014 Information Processing Society of Japan

者が多いならば，C 言語の配列の添え字は 0 から始まるので繰り返しの終了条件に気をつける，といったアドバイスを全体に与える．問題文には記述されていない，配列の要素の取りうる値の範囲を 100 以下と仮定して次のように記述している学習者が多い場合は，100 より大きい数値の入力もあり得ることを全体に説明することで，作成するプログラムの仕様について再考させることができる．

```
int min=101;
for(i=0;i<size;i++) {
  if(d[i]<min) min=d[i];
}
return min;
```

次の記述は模範解答とは異なっているが，これは誤りではなく別解であり，学習者に指導をしなくてもよい．

```
int i, minpos = 0;
for(i = 1; i < size; i++)
   if(d[i] < d[minpos]) minpos = i;
return d[minpos];
```

教員が教室内を巡回して，学習者からの質問に応じたり，各学習者の編集画面を注視したりすることで，コーディング状況をある程度把握することは可能であるが，演習時間には限りがあり，各学習者に割ける時間が限られてしまう．TA を活用した場合でも，学習者のコーディング状況は TA 毎に把握されるが，教員と TA の間で学習者の状況が共有されずに，複数人に同じようなアドバイスが必要であるにもかかわらず，学習者毎に個別に対応する状況が起こることがある．上述した演習問題の例のように，教員が用意した模範解答とは異なった記述をしている学習者がどの程度いるか，異なった記述は具体的にどのような記述かなどを教員が巡回を行う方法以外で確認できるようになれば，コーディング状況が把握できたと言える．教育支援システムとして支援するためには，学習者のソースコードの字句や構文などを解析する必要があるが，学習者によってプログラムの書き方が異なるので，単純に解析しても同じような記述が集約されずに，学習者全体の傾向がわからないことがある．

本研究では，演習時間中の学習者のソースコードを定期的に取得し，学習者毎の差異を吸収するために演算子，式，文，文の構造という観点で分析することで，コーディング状況が把握可能かを考察する．ソースコードの分析には，サイクロマチック複雑度などのソフトウェアメトリクスを用いる方法や，模範解答と学習者のソースコードの抽象構文木や PDG(Program Dependency Graph) を用いた類似度の比較などの方法も考えられる．本研究ではコンパイルができない作成途中のソースコードも分析の対象とするので，構文解析やフロー解析が行えない場合もある．そこで，演算子や制御文とその条件式などの特徴的な箇所を，字句を基に抽出して分析を行う方法を採用し，その結果について調査を行った．

ソースコードを取得するために Web を用いた統合開発環境 (Web Integrated Development Environment, WebIDE) を実現した．一般に用いられている Eclipse などの統合開発環境や既存のエディタなどでもプログラムの自動取得は可能であるが，そのためには，各学習者が利用する PC で環境構築や更新作業などが必要となる．WebIDE ではサーバ上での環境構築や設定が必要であるが，クライアント PC での設定作業の手間が省けるという利点がある．

25 人の学習者に演習問題を 4 題出題し，学習者の作成途中のソースコードを随時取得して，手動で分析を行ったところ，提案した方法でコーディング状況の把握が可能であることを確認した．

2. 関連研究

プログラミング演習において，学習者の進捗状況を把握するための教育支援システムはいくつか提案されている．

文献 [1] は統合リアルタイム授業支援システム (IDISS: Integrated and Dynamic Instruction Support System) により，授業中にリアルタイムに課題を提示し，それに対し学習者が提出したソースコードについて，コンパイルやインデントなどを自動評価することができる．また，学習者の提出履歴を収集して分析を行うことができる．これにより，多くの学習者を少数の指導者で担当するという人的リソース問題を解消できるほか，リアルタイムで課題を作成しつつ理解度に応じた授業進行が可能となる．

文献 [2] は，C3PV(Coding Process Visualizer in Programming Practice) を用いて学習者のコーディング過程を分析し，コーディング遅延や停滞を可視化している．学習者毎に，総 LOC（Lines Of Code）や課題毎のコーディング時間，単位時間あたりのエディタ操作数，課題毎のエラー継続時間といった 4 種類のコーディング過程メトリクスをリアルタイムに計測し，ランキング等の可視化処理を行い，教員に提示する．C3PV が相対的に状況が悪い学習者をサポートが必要な対象として提示することで，限られた数の教員や TA でも学習状況を把握でき，その状況に即した指導が可能となる．

文献 [3] ではプログラミング初学者のコーディング過程を分析し，その傾向を明らかにしている．初学者を対象としたプログラミング演習時に，ファイル保存時やコンパイル時などのタイミングでソースコードを取得し，これらを対象として定性的および定量的な分析を行っている．定性的な分析としては，コンパイル時のエラーメッセージとその際のソースプログラムを基に，初学者が課題の要求に沿った編集を行っていたかをプログラミング熟練者が目視による分析を行っている．定量的な分析としては，ソースコードのトークンに基づく編集距離に着目し，その値の傾向を分析している．

文献 [4] では，学習者にプログラムの動作を理解をさせ

ⓒ 2014 Information Processing Society of Japan

るためのアニメーションシステムを用い，その操作履歴から動作理解困難箇所を集計して教員へ通知する．また，学習者のコンパイルエラーの傾向を分析し，エラーの原因や場所も教員へ提示する．これによりリアルタイムに学習者全体の問題傾向が把握でき，共通の問題を抱える学習者に対して一斉指導を行うことを可能にしている．

既存の研究では学習者の進捗状況について，他の学習者と相対的に見て作業が遅れている学習者や，模範解答とは異なる記述をしている学習者の存在を把握できる．しかし，学習者がどのような記述をしているのかを把握することはできない．

本研究では，作成途中のソースコードに対して分析を行うことで，学習者のコーディング状況を把握する方法について考察する．

3. Web 統合開発環境によるソースコード取得

プログラミング演習において，学習者全員のコーディング状況を分析しようとする場合，学習者のソースコードを一箇所に集める必要がある．本研究では，Web を用いて開発環境を実現することで，ソースコードの自動取得および集約を行う．

3.1 基本機能

本研究では，学習者の進捗状況を把握するためのプラットフォームとして，WebIDE を実現する．プログラミングを学ぶ上で必要な機能（以下，基本機能）に加え，定期的に作成途中のソースコードを取得する機能を実現した．WebIDE の基本機能は次の通りである．

- アカウント管理機能
  学習者が WebIDE を利用するために，アカウント管理用データベースを用い，ログイン認証を行う．
- ファイル管理機能
  各学習者が作成したファイルをサーバに学習者ごとに保存する．ファイルに対して行うことができる操作としては，オープン，保存があり，いずれも WebIDE 上で行うことができる．
- コンパイル・実行機能
  学習者が記述したソースファイルをコンパイル・実行し，WebIDE 上に実行結果の表示を行う．

非機能要求としては，特定のプログラミング言語に依存しないことや，操作性や処理速度などにおいて，ローカル環境と比べても学習に支障がないシステムであることが挙げられる．

3.2 ソースコードのコンパイル・実行

ソースコードのコンパイル・実行には，Ideone.com[5] の WebAPI を利用した．Ideone は 40 種以上のプログラミング言語に対応したオンラインコンパイラであり，ソースコードを送信することで，コンパイルおよび実行結果を受け取ることができるサービスである．数十行程度のプログラムであれば平均 4〜5 秒でコンパイル・実行とその結果を WebIDE に表示することができ，学習をするにあたって，大きな支障のない速度で処理が行える．また，無限ループするプログラムに対しては実行を強制終了させるなどの処理も考慮されている．

図 1 WebIDE の画面例

3.3 実現

WebIDE の実際の画面を**図 1** に示す．WebIDE のプログラムの規模は，HTML 149 行，PHP 248 行，JavaScript 197 行である．

本研究で実現した WebIDE では標準入出力を扱えるが，fopen 関数などを利用したファイルの入出力，ソケットを利用して通信を行うプログラム，system 関数やシステムコールを利用するプログラム，学習者のローカル PC に新規にウィンドウを作成してグラフィックスを表示させるプログラムの実行などには対応していない．

4. コーディング状況把握方法

本研究では，プログラミング演習において学習者がソースコードをどのように記述しているか，学習者全体でどのような記述があるのかを分析することにより，学習者のコーディング状況の把握を目指す．

演習の出題形式については，教員がプログラムの雛形と模範解答プログラムを予め用意しておくことを前提とする．雛形には教員が学習意図を考慮して関数のプロトタイプなどを定義しておき，関数の本体などを学習者が記述するものとする．模範解答プログラムは技巧的なプログラムではなく，学習において学んだ構文などを用いた標準的なプログラムとする．

用意した模範解答と学習者のソースコードを比較することで，学習者がどの程度模範解答に近い記述をしているか，

ⓒ 2014 Information Processing Society of Japan

別のプログラムを記述しているかというコーディング状況の分析を行う．ソースコードは同じ処理を記述していたとしても，式や文の記述方法は学習者毎に異なる場合がある．出現する字句の記述人数や記述回数を単純に数えることは可能であるが，それらを教員に掲示してもコーディング状況をすぐに把握することは困難である．学習者による記述の違い，演習において学習者に理解してもらいたい重要箇所を考慮して，本研究では，字句，式，文，文の構造の 4 つの観点からの分析方法を考えた．

1. **字句**　模範解答に出現する字句が記述できているかを分析する．4.1 節で述べるように，本研究では演算子に限定して分析する．
2. **式**　演算子やリテラルなどを用いて，条件式などが記述できているかを分析する．
3. **文**　プログラムの制御構造として重要である条件分岐や繰り返しの文などが，模範解答に出現する条件式を用いて記述できているか分析する．
4. **文の構造**　制御文などの入れ子関係が記述できているかを分析する．

学習者全体のコーディング状況を把握したいので，分析対象の記述とその記述人数，出現回数を調べる．

4.1 演算子による分析

学習者のソースコードと模範解答プログラムの比較を行う方法として，記述された字句やその出現回数を調べることが考えられる．字句には if や for などの制御文の予約語や，int, double などの型名，変数名，=, +, <= などの演算子，セミコロン ; や括弧 { } などの区切り記号などが挙げられるが，すべての字句の出現回数を数えて教員に提示しても，セミコロン; などの回数が多くなり，コーディング状況の把握には適さない．

本研究では，一般に模範解答と同じ計算を行うには，模範解答に出現する演算子を用いる必要があることから，演算子に着目して，学習者のソースコードの分析を行う．if や for などの予約語もプログラムの構造の分析には重要であるが，これらは字句としてではなく，4.3 節で述べるように制御文として分析を行う．型は主に変数宣言において記述されるが，宣言文では複数の変数が宣言できるので，学習者のソースコードと模範解答では型の字句の出現回数が大きく異なる可能性があり，分析対象としないことにした．変数名は学習者によって異なることがあり，学習者全体の傾向をつかむのに適していないと考えた．区切り記号は構文解析を行う上で必要な字句であり，必ずしも模範解答と同じ区切り ({} の付け方など) をする必要がないと考えた[*1]．

[*1] 演習で if や while では，本体が単文でも必ず {} で囲むように徹底させる場合などはこの限りではないが，今回は対象としない．

4.2 特徴式による分析

学習者が模範解答に出現する演算子を記述していたとしても，式として適切に記述できているとは限らないので，式の単位でも分析を行う．式の記述方法は学習者によって異なる場合があり，すべての式を分析の対象とすると，記述者が少ない式が多く現れ，コーディング状況がわかりづらくなる恐れがある．

本研究では，プログラムの制御構造に関連する条件分岐や繰り返しなどの制御構文における条件式，および，関数の本体を記述する演習問題を考えて，関数の計算結果である return 文における式を，プログラムにおける重要な式として分析対象とする．以降，これらの式を**特徴式**と呼ぶ．

特徴式を構成する部分式には変数参照式も含まれる．プログラムの雛形に現れる関数の仮引数名などは，模範解答にも学習者のソースコードにも現れるので分析の対象とする．学習者が定義した変数は学習者によって変数名が異なることがあるので，学習者が定義した変数名は特徴式の分析の対象としない．

0 や'\0' などのリテラルは模範解答と学習者の特徴式に含まれる．リテラルは条件の境界値を表していることが多く，これらを適切に記述できるかはコーディング状況の把握に有効であると考え，分析の対象とする．例えば，1 節のプログラム例では，`<size`, `d[]<` などが特徴式となる．

4.3 特徴文による分析

学習者が模範解答に出現する特徴式を記述していたとしても，それを適切な箇所，例えば，繰り返しの条件など，に記述できているとは限らない．プログラムにおいて，制御構造が適切に記述できるかは重要である．本研究では，条件分岐や繰り返しなどの制御文と関数の計算結果を表す return 文を文の単位での分析対象とする．以降，これらを**特徴文**と呼ぶ．特徴文の分析においては，制御文と条件式の組み合わせ，return 文と返却式の組み合わせを分析する．つまり，特徴式を用いて文を記述できているかを分析する．例えば，1 節のプログラム例では，`for (<size)`, `if (d[]<)` などが特徴文となる．

4.4 特徴文の構造による分析

ある程度の規模のプログラムでは，条件分岐や繰り返しの入れ子が現れる．学習者が制御フローを理解し，これらを適切に記述できることは重要であるので，特徴文の入れ子関係を**特徴文の構造**として分析を行う．深さが 2 以上の入れ子は，複数の入れ子の組み合わせと考えて，本研究では深さが 1 の特徴文の入れ子関係を分析対象とする．

5. 実験と評価

5.1 実験方法

C 言語を一通り学習した学部の 3, 4 年生 25 名に演習課

ⓒ 2014 Information Processing Society of Japan

題を出題し，本研究で実現した WebIDE を利用してもらい，1 分ごとに学習者のソースコードを取得した．学習者のソースコードについて，4 節で述べた分析方法を目視で適用し，コーディング状況が把握できるかを確認する．

学習者には関数を作成する次の 4 問を出題した．

(1) 文字列 from を to にコピーする関数
`void strcopy(char *from, char *to)`

(2) 実数 a と整数 n を受け取り，a の n 乗を計算する関数
`double power(double a, int n)`

(3) 文字列 str の長さを返す再帰関数
`int countStrLength(char *str)`

(4) 文字列 str から指定された文字 c を削除する関数
`void strdelete(char *str, char c)`

出題において，作成する関数の本体部分が空欄になったプログラムを雛形として学習者に提示した．main 関数はテストドライバとして記述し，出力結果も提示している (**図 2**)．分析は作成する関数の本体部分 (ブロックの中身) を対象にした．

```
/* ex0001 */

#include <stdio.h>

/* 文字列 from を to にコピーする */
void strcopy(char *from, char *to)
{
    //ここに解答を記述
}

int main(void)
{
    char s1[10] = "abc";
    char s2[10] = "xyz";
    char s3[10] = "ab";

    strcopy(s1, s2);
    printf("%s\n",s1);
    printf("%s\n",s2);
    strcopy(s3, s1);
    printf("%s\n",s1);
    printf("%s\n",s2);

    return 0;
}
/*  実行結果
abc
abc
ab
abc
*/
```

図 2　問題 1 の出題形式

各問題の模範解答例を**図 3**，**図 4**，**図 5**，**図 6** に示す．

問題 1 は繰り返し文が 1 つで，プログラムの構造が比較的単純であるので，本稿では，演算子の分析によるコーディング状況把握について確認する．問題 2 は n の正負の場合分けと累乗計算のための繰り返しが必要になり，別解もいくつか考えられる．例えば，$n < 0$ の場合に a^{-n} を求めてから，$1/a^{-n}$ するなどが別解として挙げられる．特徴式，特徴文の分析により，別解も含めてコーディング状況が把握できるかを確認する．問題 3 は再帰の問題であり，

```
void strcopy(char *from, char *to)
{
    for (; *from != '\0'; from++, to++) {
        *to = *from;
    }
    *to = '\0';
}
```

図 3　問題 1 の模範解答例

```
double power(double a, int n)
{
    double ans = 1;
    if (n >= 0) {
        for(; n > 0; n--) {
            ans *= a;
        }
    } else {
        for(; n < 0; n++) {
            ans /= a;
        }
    }
    return ans;
}
```

図 4　問題 2 の模範解答例

```
int countStrLength(char *str)
{
    if (*str == '\0') {
       return 0;
    } else {
       return 1 + countStrLength(str + 1);
    }
}
```

図 5　問題 3 の模範解答例

```
void strdelete(char *str, char c) {
    char *t;
    for (t = str; *str != '\0'; str++) {
        if (*str != c) {
            *t = *str;
            t++;
        }
    }
    *t = '\0';
}
```

図 6　問題 4 の模範解答例

表 1　問題 1 における学習者の演算子の記述

5 分時点		10 分時点		15 分時点	
演算子	人数 (回数)	演算子	人数 (回数)	演算子	人数 (回数)
=	13 (30)	=	22 (43)	=	23 (55)
++	11 (17)	++	17 (29)	++	18 (38)
!=	8 (9)	!=	16 (16)	!=	17 (17)
==	1 (2)	==	1 (1)	==	3 (3)
<	1 (1)	<	1 (1)	<	2 (2)
				+=	1 (1)

if と return の組み合わせが適切に記述できているかが重要であるので，特徴文の構造による分析結果について述べる．問題 4 は文字列の走査，2 つのポインタの走査などが組み合わさった総合的な問題として，演算子，特徴式，特徴文，特徴文の構造による分析でコーディング状況が把握できるかを確認する．

5.2　演算子による分析

問題 1 について，模範解答に出現する !=, ++, = の記

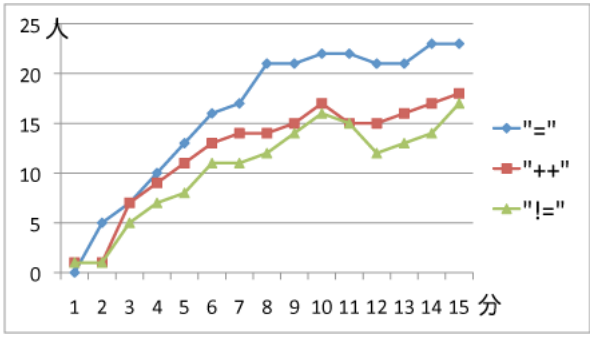

図 7 問題 1 における模範解答に出現する演算子の記述人数

述人数のグラフを**図 7** に，5 分後，10 分後，15 分後の全学習者のソースコードに記述されていた演算子を**表 1** に示す．表 1 では記述人数の多い順に並べている (以降の表も同様である)．時間の経過にともない，模範解答に出現する = や ++, != の演算子を記述している人数が増え，その出現回数も模範解答に近づいているのがわかる．一方で，模範解答に出現しない == や < を記述している学習者がいることもわかり，演算子により学習者のおおまかなコーディング状況を把握することができた．

なお，演算子に限定せず，すべての字句の記述人数を数えると 10 分時点で，; と = が 22 人，(と) が 19 人，from が 18 人，++ が 17 人，!=が 16 人，'\0', to, } が 15 人となった．区切り記号が多く現れ，直感的に状況が把握しづらいので，演算子を抽出した分析は有効であると言える．

5.3 特徴式による分析

問題 2 について，5 分後，15 分後の学習者が記述した特徴式とその人数を**表 2** に示す．模範解答に出現する `n>=0` の記述が時間経過にともない，増加しているのがわかる．記述人数の多い `<n` や，`>0` は模範解答には現れないので，別解の記述が推測されるが，これらの式が条件分岐に書かれたものか，繰り返しに書かれたものかがわからないので，これだけでコーディング状況を判断するのは難しい．また，`*power(,n-1)` から再帰で計算している学習者がいることが推測される．

5.4 特徴文による分析

問題 2 について，特徴文の分析結果を**表 3** に示す．return 文の特徴式が空なのは，学習者が定義した変数を記述しているからである．for と `<n` の組み合わせが多いことから，n が正の場合の累乗計算を `for (i=0; i<n; i++)` のように記述している学習者多いと推測される．if 文の特徴式に n と 0 との比較が出現することから，n の場合分けも記述できていると考えられる．return と 1/ により，$1/a^{-n}$ で答を求めている学者者の存在がわかる．return における*power(,n-1) などの記述の存在から，再帰による別解を作成している学習者がいることも把握できる．

表 2 問題 2 における学習者の特徴式の記述

5 分時点		15 分時点	
特徴式	人数	特徴式	人数
<n	13	<n	17
>0	6	>0	14
n>=0	3	n>=0	6
n>0	3	>n	6
0	2	n<0	5
n==0	2	n==0	5
1	1	1	3
*power(,n-1)	1	<=n	3
*	1	1/	3
<=	1	*power(,n-1)	1
<=n	1	!=0	1
<n-1	1	(1/)*power(,n+1)	1
>n+1	1	>=n	1
1/	1	n<-1	1
n<0	1	n<1	1
n==1	1	n>0	1

表 3 問題 2 における学習者の特徴文の記述

5 分時点			15 分時点		
特徴文	特徴式	人数	特徴文	特徴式	人数
return		16	return		23
for	<n	13	for	<n	17
if	>0	5	if	>0	13
for	n>0	3	for	>n	6
if	n>=0	2	if	n>=0	6
if	n==0	2	if	n==0	5
return	0	1	if	<=n	3
for	<=n	1	if	n<0	3
for	<n-1	1	return	1/	3
for	>n+1	1	return	1	3
if	n<0	1	for	>=n	1
if	n==1	1	for	n<-1	1
return	*power(,n-1)	1	for	n<0	1
return	1/	1	for	n>0	1
return	1	1	for	n<1	1
for	>0	1	if	>0	1
for	<=	1	return	(1/)*power(,n+1)	1
return	*	1	return	*power(,n-1)	1
			while	!=0	1
			while	n<0	1
			while	n>0	1

特徴文の分析により，学習者のおおまかなコーディング状況を把握することができ，模範解答とは異なる別解を記述している学習者の存在も確認できた．

5.5 特徴文の構造による分析

問題 3 について，5 分後，15 分後の特徴文の構造を**表 4**，**表 5** に示す．なお，表には 2 人以上記述していたコードを載せている．if(*str==’\0’) と return 0 の組み合わせは，時間経過毎に増加しているが，return 0 でなく return 1 を記述している学習者が数名いることから，文字列の長さが正しくカウントされない，または別解を作成していることが把握できる．全体的に記述人数が少ないので，多くの学

ⓒ 2014 Information Processing Society of Japan

表 4 問題 3 における 5 分時点で特徴文の構造

組み合わせ		人数
if(*str==’\0’)	return 1	2

表 5 問題 3 における 15 分時点で特徴文の構造

組み合わせ		人数
if(*str == ’\0’)	return 0	5
else	return 1+countStrLength(str)	3
if(*str == ’\0’)	return 1	3
else	return	2
else	return 0	2

表 6 問題 4 における学習者の演算子の記述

10 分時点		20 分時点		30 分時点	
字句	人 (回数)	字句	人数 (回数)	字句	人数 (回数)
=	19(72)	=	19(90)	=	18(81)
++	19(37)	!=	18(28)	!=	18(30)
!=	18(23)	++	18(47)	++	18(50)
==	15(16)	==	17(21)	==	15(16)
+	1(3)	–	3(4)	–	2(6)
&&	1(1)	+	2(3)	<	1(2)

習者が文字列の再帰処理が記述できていないことがわかる．実際の演習では，学習者全員に文字列の再帰的定義を示す，などのヒントを与えるとよい．

5.6 総合的な分析

問題 4 に，演算子，特徴式，特徴文，特徴文の構造による分析を行い，把握できたコーディング状況について述べる．問題 4 は学習者が解答を作成する時間が他の問題に比べて長かったので，10 分時点，20 分時点，30 分時点のソースコードを分析した．

5.6.1 演算子による分析

演算子による分析結果を**表 6** に示す．この問題では，模範解答で登場している演算子を学習者が多く記述している．= と ++ 演算子は人数と回数の両方が多いが，!= 演算子は人数に対しての回数が少ない．解答例にない == 演算子を多くの学習者が使用していることから，!= ではなく，== を比較に用いていることが把握できる．出現する演算子の種類と回数は 10 分以降で，大きな変化はないことから，学習者は式，文の記述でつまずいていると推測される．

5.6.2 特徴式による分析

特徴式の分析結果を**表 7** に示す．模範解答例にある条件式 *str!=’\0’ は，学習者でも多くの人数が記述している．*str!=c に関しては，使用している人数は時間ごとに徐々に増加しているが，記述している学習者は多くない．解答例には使われていない条件式 *str==c は 10 分時点から 30 分時点で記述人数が減少している．学習者が *str==c から徐々に *str!=c に書き換えていることが把握できる．

5.6.3 特徴文による分析

特徴文の分析結果を**表 8** に示す．解答例にある制御文 for と*str!=’\0’，および，同様の while と*str!=’\0’ については，10 分時点から 20 分時点で減少し，30 分時点でまた増加している．学習者の記述で for と!=’\0’ を確認すると，10 分時点から 20 分時点で増加し，30 分時点で減少している．学習者は for 文の条件において，*str の代わりに自身が宣言した変数を使用して，プログラムを記述していることが把握できる．模範解答例の if と *str!=c の組み合わせについては，時間経過毎に徐々に増加している．学習者の if と *str==c の記述が経過 10 分と 20 分時点で多い．30 分時点で減少しているが，if と ==c の記述が増えているので，学習者は独自の変数を定義していることがわかる．

表 7 問題 4 における学習者の特徴式の記述

10 分時点		20 分時点		30 分時点	
*str!=’\0’	15	*str!=’\0’	13	*str!=’\0’	16
*str==c	11	*str==c	12	*str==c	8
!=’\0’	3	!=’\0’	6	*str!=c	7
*str!=c	2	*str!=c	6	!=’\0’	6
==c	2	==c	3	==c	4

表 8 問題 4 における学習者の特徴文の記述

10 分時点			20 分時点			30 分時点		
特徴文	特徴式	人数	特徴文	特徴式	人数	特徴文	特徴式	人数
if	*str==c	11	if	*str==c	11	for	*str!=’\0’	8
for	*str!=’\0’	9	while	*str!=’\0’	7	if	*str==c	8
while	*str!=’\0’	6	for	*str!=’\0’	6	if	*str!=c	7
for	!=’\0’	3	for	!=’\0’	6	while	*str!=’\0’	7
if	*str!=c	2	if	*str!=c	6	if	==c	3

5.6.4 特徴文の構造による分析

特徴文の構造の分析結果を**表 9** に示す．解答例の for(*str!=’\0’) と if(*str!=c)，および，同様の while(*str!=’\0’) と if(*str!=c) の組み合わせは一部の学習者のみが記述している．30 分時点で，if(*str!=c) の記述は 7 人いたが，for, while と組み合わせた記述は 4 人であり，正しく記述できていないことが把握できる．全体を通して for(*str!=’\0’) と if(*str==c) の組み合わせが多いことがわかる．

5.6.5 分析のまとめ

問題 4 ではプログラム作成開始後 10 分程度で，プログラム作成に必要な演算子は記述できているが，制御文の組み合わせが適切に記述できていない状況がわかった．特徴式の分析から文字列の比較の条件式を記述しているが，特徴文とその構造の分析結果から，文字列を最後まで走査する繰り返しは書けても，その中で文字列から削除する文字を判定する条件分岐が書けていない学習者が多いことがわかった．

実際に学習者のソースコードを確認すると，模範解答の t に相当するポインタ変数を定義せずに，仮引数の str のみで処理しようとしていた者が多かったが，今回の分析方法では検出できなかった．学習者が定義した変数の個数や型情報についての分析方法も，今後検討する必要がある．

ⓒ 2014 Information Processing Society of Japan

表 9 問題 4 における学習者の制御文の構造の記述

10 分時点		
組み合わせ		人数
for(*str!='\0')	if(*str==c)	6
while(*str!='\0')	if(*str==c)	3
for(!='\0')	if(==c)	2
while(*str!=NULL)	if(str==c)	1
for(*str!='\0')	if(*str!=c)	1
20 分時点		
組み合わせ		人数
for(*str!='\0')	if(*str==c)	6
while(*str!='\0')	if(*str==c)	4
for(*str!='\0')	if(*str!=c)	3
for(!='\0')	if(==c)	2
while(*str!='\0')	if(*str!=c)	1
30 分時点		
組み合わせ		人数
for(*str!='\0')	if(*str==c)	6
while(*str!='\0')	if(*str==c)	2
while(*str!='\0')	if(*str!=c)	2
for(*str!='\0')	if(*str!=c)	2
while(*str!='\0')	if(str=='c')	1

5.7 教員への提示方法

演算子や特徴式，特徴文について分析することで，学習者の誤りや別解の記述も含んだコーディング状況をある程度把握可能であることがわかった．しかし，これまでに示した表形式のように，演算子や特徴文とそれを記述している人数を示すだけでは，教員が一目でコーディング状況を把握することは難しい．

学習者が記述しているコードとその記述人数を**図 8** のようなグラフで，定期的に，例えば 1 分毎に更新して教員に掲示することでコーディング状況が把握しやすくなると期待できる．図 8 は問題 4 の特徴文の構造による分析結果の 10 分時点と，20 分時点の様子を表したグラフであり，色の濃い棒グラフが模範解答にある記述である．制御文の入れ子を記述できている学習者が少なく，記述している場合でも模範解答とは異なる記述が多いことがわかる．学習者全員に対してヒントを出すなどの指導を行うかは教員の判断に委ねられるが，グラフはその判断材料になる．

6. おわりに

本研究では，学習者のコーディング状況を把握するために，ソースコードを演算子，特徴式，特徴文，特徴文の構造の 4 つの観点から分析を行う方法について考察した．WebIDE を用いて学習者から作成途中のソースコードを取得し，提案した方法で，誤った記述や別解を検出することができ，コーディング状況の把握が可能であることを確認した．

今後の課題として，より精度の高い分析方法の考察が挙げられる．5.6.5 節で述べたように，学習者が定義した変数の個数や型情報の分析方法について検討する必要がある．

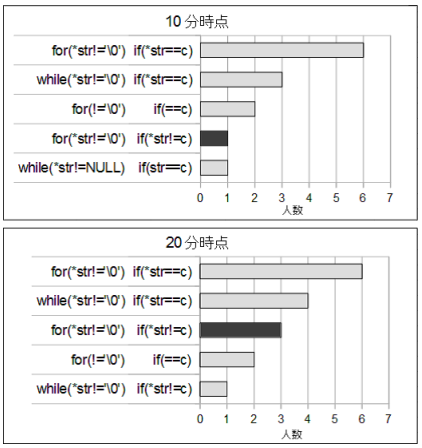

図 8 問題 4 の特徴文の構造のグラフ

関数の作成においては，仮引数を用いた記述の分析もコーディング状況の把握に有効である．今回の方法では制御文の条件式や return の式に仮引数が出現する場合は分析するが，それ以外にも仮引数が出現する代入文の右辺などの計算式を模範解答と比較することで，仮引数を適切に使用できているかを把握できると考える．

WebIDE で実際にリアルタイムに学習者のソースコードを分析して教員に通知する機能の実現も今後の課題である．作成途中のソースコードはコンパイルができない場合があるが，コード断片の解析が可能な TEBA などの構文解析器 [6] を用いることで，解析可能であると考える．

謝辞

WebIDE の開発，実験の実施，学習者のプログラムの分析などに協力していただいた南山大学情報理工学部卒業生の松井暁生君，石川航君，浅井祐太君に感謝します．

本研究の一部は，JSPS 科研費 26350344, 2014 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] 長谷川伸，松田承一，高野辰之，宮川　治：プログラミング入門教育を対象としたリアルタイム授業支援システム，情報処理学会論文誌，Vol. 52, No. 12, pp. 3135–3149 (2011).
[2] 井垣　宏，齊藤　俊，井上亮文，中村亮太，楠本真二：プログラミング演習における進捗状況把握のためのコーディング過程可視化システム C3PV の提案，情報処理学会論文誌，Vol. 54, No. 1, pp. 330–339 (2013).
[3] 伏田享平，玉田春昭，井垣　宏，藤原賢二，吉田則裕：プログラミング演習における初学者を対象としたコーディング傾向の分析，電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス，Vol. 111, No. 481, pp. 67–72 (2012).
[4] 倉澤邦美，鈴木恵介，飯島正也，横山節雄，宮寺庸造：プログラミング演習における一斉指導のための学習状況把握支援システムの開発，電子情報通信学会技術研究報告. ET, 教育工学，Vol. 104, No. 703, pp. 19–24 (2005).
[5] Sphere Research Labs: *Ideone.com - Online Compiler and IDE*. http://ideone.com/.
[6] 吉田　敦，蜂巣吉成，et al.：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).

ⓒ 2014 Information Processing Society of Japan