プログラミング演習における構文要素の種類毎のビューによるコーディング状況把握方法の提案

石元 慎太郎[1,a)]　蜂巣 吉成[2,b)]　吉田 敦[2,c)]　桑原 寛明[2,d)]　阿草 清滋[3,e)]

概要：プログラミング演習において，学習者の編集途中のソースコードを分析してコーディング状況を把握する方法を提案する．C 言語を対象に，条件分岐や繰返しなどのプログラムの制御文を抽出してコーディング状況を大まかに把握し，代入文における演算子や変数の型などを見ることでコーディング状況を細かく把握する方法を提案する．十数名に対する演習に提案方法を用いたところ，学習者の誤っているソースコード記述がわかることを確認した．

キーワード：プログラミング教育，コーディング状況，構文要素

Assessing Coding Situations on a Programming Exercise from Views of Syntax Element Types

Shintaro Ishimoto[1,a)]　Yoshinari Hachisu[2,b)]　Atsushi Yoshida[2,c)]　Hiroaki Kuwabara[2,d)]
Kiyoshi Agusa[3,e)]

Abstract: We propose a way of assessing learners' coding situations on programming exercise by analyzing their editing source codes. We can understand their codes roughly by extracting branches and loops and finely by extracting operators and types of variables on assignments. We have confirmed that we can find errors of students' editing codes through programming exercises of eleven students.

Keywords: Programming Exercises, Coding Situations, Syntax Elements

1. はじめに

現在，大学で行われているプログラミング教育では，学生数十名に対して，教員 1 人，数名の TA による演習形式で授業が行われていることが多い．効率よく指導を行うためには，教員や TA が学生のソースコード記述の内容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，実数 x の n 乗を求める課題を考える．型についてよく理解せずに，整数型の変数に計算結果を代入している学習者が多いときや，n が 0 以上かどうかの場合分けが書けていない学習者が多ければ，それらについて全体指導をするとよい．

教員や TA が学生からの質問に答えたり，教室内を巡回して編集中のソースコードを確認することで，コーディング状況を把握することは可能であるが，限られた演習時間の中で，学習者全員の傾向をつかむのは難しい．

解決策として進捗状況把握システムを利用する方法が挙げられる．長島らはコンパイルエラーや実行結果などで進捗状況を把握する方法を提案しているが [6]，どのようなソースコードを記述しているかまではわからない．

文献 [1], [2], [3] ではソースコードの行数，コンパイル状

[1] 南山大学 大学院理工学研究科 Graduate School of Science and Engineering, Nanzan University, 18 Yamazato-cho, Showa-ku, Nagoya-shi, 466-8673, Japan
[2] 南山大学 Nanzan University, 18 Yamazato-cho,Showa-ku, Nagoya-shi, 466-8673, Japan
[3] 京都高度技術研究所 ASTEM, 134, Chudouji Minamimachi, Shimogyou, Kyoto, 600-8813, Japan
[a)] m18se004@nanzan-u.ac.jp
[b)] hachisu@nanzan-u.ac.jp
[c)] atsu@nanzan-u.ac.jp
[d)] kuwabara@nanzan-u.ac.jp
[e)] agusa@astem.or.jp

況，インデントといった要素に着目した進捗状況把握手法を提案している．しかし，これらの研究は学生がコンパイルをしたソースコードに対して分析を行うものであり，編集途中やエラーとなるソースコードに対しては進捗状況の把握が行えない．

われわれはすでに，C 言語を対象に編集途中のソースコードにおける制御文や演算子の出現回数などからコーディング状況を把握する方法を提案した [5]．さらに，制御文の入れ子関係などの制御構造と条件式に着目してコーディング状況を把握する方法を提案した [4]．制御構造に基づいて同値関係を定義し，学生のソースコードを同値類分割することで，教員が確認すべきソースコード数を減らすことができ，学生全体のコーディング状況の把握が容易になる．しかし，条件式でも同様に同値関係を定義して同値類分割をすると分割数が多くなることや，制御構造と条件式だけではコーディング状況を把握できない場合があった．

本研究では，[4] の研究を発展させ，制御構造に加えて，代入文における演算子や変数の型などを用いてコーディング状況を把握する方法を提案する．手続き型言語では代入によりプログラムの状態を変化させながら計算が進むので，コーディング状況を把握する上で代入は重要であると考えた．変数名などは学習者毎に異なるので，変数の型として抽出する．初学者にありがちな型の誤りなども把握できる．本研究でも複数の同値類分割を用いるが，その結果をビューと呼ぶ．制御文のビューではプログラムの制御構造，処理の流れがわかり，コーディング状況を大まかに把握できる．制御文と演算子，型を見る演算子と型のビューでは演算の過程がわかり，細かなコーディング状況が把握可能となる．

十数名に対する演習に提案方法を用いたところ，制御文のビューでは不要な制御文に関する間違いを確認できた．演算子と型のビューでは制御文のビューでは確認できなかった再帰関数の return が書けていない，演算子の記述漏れ，ポインタの利用方法といった間違いを確認できた．

2. 関連研究

プログラミング演習において学生の進捗状況を把握するためのシステムはいくつか提案されている．

井垣ら [1] は学生のコーディング過程を分析し，可視化して教員へ提示するシステム C3PV を提案している．学生が入力したソースコードの行数，課題ごとのコーディング時間，単位時間あたりのエディタ操作数，エラー継続時間の 4 種類のメトリクスを計測し，学生全体の進捗を比較する．相対的に遅れている学生に対し教員や TA が指導することによって授業の改善を行っている．

長谷川ら [2] は統合リアルタイム授業支援システム IDISS を提案している．リアルタイムに課題を提示し，提出された学生の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で評価し通知することができる．また，学生の提出履歴を収集し分析することもできる．これにより人的リソースの問題を解消でき，リアルタイムで課題を作成し理解度に応じた授業を行うことが可能となる．

伏田ら [3] は学生が誤りに陥る際の傾向を明らかにすることを目的とし，コーディング過程を分析している．分析にあたっては，ファイル保存時やコンパイル時などにソースコードを取得し，そのソースコードを基に定性的及び定量的な分析を行っている．定性的な分析としてはプログラミング熟練者による目視を行い，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傾向を明らかなものとしている．以上の分析結果から学生の進捗状況を把握し，授業の改善を行っている．

長島ら [6] はブラウザ上でソースコードを取得できる初学者向けプログラミング環境 Bit Arrow を用いた教員支援機能を提案している．教員が学習者の作業の状況を把握する方法として学習者ごとにプログラムのエラーと実行回数，エラーで作業が止まっている時間，現在行っている問題を示すことで教員へ学生の作業状況を示す．

これらの研究では，相対的に進捗が遅れている学生や誤ったソースコードを記述している学生の特定をすることはできるが，コーディング過程やどのような誤りをしているかを把握することは難しい．また，コンパイル時やファイル保存時などのソースコードを参考にするので，任意のタイミングでのソースコードを見ることはできない．

われわれは，任意のタイミングでソースコードを保存・取得できる Web 上のプログラミング環境 (WebIDE) を用いて編集中のソースコードから制御文や演算子などの字句を抽出して，その出現回数を計測して，模範解答における出現回数との違いからコーディング状況を把握する方法を提案した [5]．模範解答と学習が異なった記述をしていることはわかるが，学習者のプログラム全体の構造を把握したり，どのような記述でつまずいているかを判断することは難しい．

さらに，われわれの研究室では，制御文の入れ子関係などの制御構造と条件式に着目してコーディング状況を把握する方法を提案した [4]．編集中のソースコードから，if, else, for, while の予約語とそれらの本体部分を表す波括弧 {}，および再帰関数呼出しの関数名を抽出する．ソースコード A,B から抽出した文字列が等しいとき，ソースコード A と B が同値であると定義し，この同値関係に基づいて学生のソースコードを同値類分割する．制御構造を抽出しているので，処理の流れが適切に記述できているかが把握でき，同値類分割により複数のソースコードがまとまることで，教員が確認すべきソースコード断片の量を減らすことができる．より詳細なコーディング状況を把握するために，制御文の条件式から演算子，リテラルなどを抽出して，同値類分割する方法も提案した．しかし，条件式の場

合は同値類としてまとめられるソースコードが少なく，教員による判断に時間がかかる．また，引数の値を交換するswap 関数のような制御構造を含まない場合はコーディング状況を把握できない．

3. 構文要素の種類毎のビューによるコーディング状況把握方法の提案

3.1 概略

本研究では，蟹江ら [4] の研究で行われていた制御構造による同値類分割に加え，代入文における演算子や型などでも同値類分割を行い，コーディング状況を把握する方法を提案する．コーディング状況の把握とは，プログラミング演習において教員が学生を指導するために，編集途中のソースコードについて，教員の意図した記述をしているか，あるいは，誤った記述をしているかを確認することである．

本研究における同値類分割とは，同値関係にあるソースコードを同値類としてまとめることであり，同値類分割の結果をビューと呼ぶ．それぞれの同値類に分類されたソースコードの個数を分類数，同値類の個数を分割数と呼ぶ．制御文のビューは分割数が少なく，少数のソースコードから学習者全体のコーディング状況を大まかに把握できる．演算子と型のビューでは，分割数が増えるが，より細かくコーディング状況を把握できる．

WebIDE により定期的に取得した学習者の編集中のソースコードに対して，補完，正規化，抽出の順に処理を行い，同値類分割を行う．

3.2 補完処理

任意のタイミングで取得した編集中のソースコードはコンパイルできない場合も多い．最低限の解析が行えるように，[4] の方法と同じく，編集中のソースコードに対し閉じ括弧を補完する．解析にはコード断片に対しても近似的な構文解析が可能なソースコード書き換え支援環境 TEBA[8] を用いる．

ソースコードはファイルの先頭から記述していくことが多く，開き括弧を記述した後でソースコードを取得すると閉じ括弧がない状態になる．式の記述の途中で二項演算子の 2 番目の被演算子がないなどの記述もあり得るが，TEBA では括弧の対応が取れていれば近似的に解析できる．

3.3 正規化処理

同値類分割において分割数を減らすために，次に示すような異なる記述で同じ結果となるものを統一した記述へ置き換える．

- 複合代入演算子 (+=など) を代入 (=) と演算 (+) に置き換える．
- インクリメント (++)，デクリメント (--) を代入 (=) と演算 (+1,-1) へ置き換える．
- コンマで区切られた初期化式を個別の初期化式へ置き換える．
- 制御文の単文を複合文にする (波括弧を追加する)．

Listing 1 は補完・正規化前のソースコード，Listing 2 は補完・正規化後のソースコードである．

Listing 1: 補完・正規化前のソースコード

```
int power(int a, int n){
    int i,ans=1;
    for(i = 0; i<n; i++)
        ans *= a;
```

Listing 2: 補完・正規化後のソースコード

```
int power(int a, int n){
    int i;
    int ans=1;
    for(i=0;i<n;i++){
        ans=ans*a;
}

}
```

異なる記述で同じ結果となる式は複数存在する．例えば，a+b*c や a+(b*c)，b*c+a などがある．本研究ではプログラミング演習を対象にしているが，初めてプログラミング言語を学ぶような場面では代入文の右辺に複数の演算子が出現するような問題は多くない．教科書 [9] では，205 問の例題中，代入の右辺に演算子が 1 つだけ出現する，もしくは出現しない問題は 202 問であり，ここで述べたような正規化処理で実用上大きな問題はない．

3.4 抽出処理

同値類分割を行うために必要な構文要素を抽出する．制御文を抽出して同値類分割をした結果が制御文のビューであり，以下のすべての抽出をした後に，同値類分割をした結果が演算子と型のビューである．

3.4.1 制御文の抽出

[4] で，制御構造によるコーディング状況把握が有効であることが示されている．本研究で扱う制御文は if 文，while 文，for 文である．switch 文は利用頻度が低いので対象外とした．制御文の予約語と波括弧を抽出する．

3.4.2 演算子の抽出

制御文より詳細にコーディング状況を把握するために，代入文における演算子と return 文の演算子を抽出する．代入文の右辺，もしくは return 文で関数呼出しが行われていた場合，実引数についても演算子の抽出を行う．手続き型言語では代入によりプログラムの状態を書き換えることで計算を進めるので，学習者のコーディング状況を確認するには変数の値の変化が重要である．return 文は関数の最終的な計算結果として重要である．特に再帰関数のコーディング状況を確認する際には return 文が正しく記述できて

いるか把握する必要がある．

3.4.3 型の抽出

代入文，return 文，関数呼出しの実引数で用いられる変数については，変数をその型に置き換えて抽出する．変数名は学習者によって異なるので，変数名のまま抽出すると同値類分割がまとまらなくなる．これまでの経験から，学習者は型の誤りに気付きにくい．特にポインタの問題では型の誤りが多い．ポインタは型名の前に p_を付けて抽出する．int 型ポインタであれば，p_int となる．配列の要素については角括弧と添字の型名を抽出する．int 型配列 a の i 番目の要素を示す a[i] であれば，int[int] となる．リテラルについてはそのままの形で抽出を行う．初期化や\0 の代入といった処理は値を含めて正しく記述できているか確認できる．宣言されていない変数の型は Undef とする．構造体には対応していない．

3.4.4 関数呼出しの抽出

代入文と return 文における関数呼出しを実引数を含めて抽出する．関数を作成する問題では関数の本体が記述できているかだけでなく，適切に関数を呼び出せているかも重要である．

3.5 同値関係の定義

ソースコード A，B について，補完・正規化・抽出したコード片が一致した場合，ソースコード A，B は同値であると定義し，本論文ではこの同値の関係を「同値関係」と呼ぶ．抽出は 3.4.1 節の制御文を抽出する場合（制御文のビュー）と 3.4 節のすべてを抽出する場合（演算子と型のビュー) がある．コードクローン [7] となるソースコードに対して同値関係を考えることもできるが，本研究では 3.4.4 節で述べたように演算子や変数の型などコーディング状況把握に有効と考える構文要素を抽出したものに対して同値関係を考えた．

実数 x の n 乗を返す関数 power の同値関係の例を示す．Listing 3，4，5，6 を補完，正規化，3.4.1 節の制御文を抽出したコード片はすべて Listing 7 となり，元の Listing 3，4，5，6 は相互に同値関係にある．

Listing 3，4，5，6 を補完，正規化，3.4 節のすべてを抽出したコード片がそれぞれ Listing 8，9，10，11 である．Listing 8，9 が一致することから，元の Listing 3，4 は同値関係にある．Listing 5，6 についても一見同値関係にあるように見えるが，抽出後の Listing 10，11 が一致しないことから Listing 5，6 は同値関係にないことがわかる.Listing 5 は初期化式が無く，Listing 6 は型の間違いがあることが抽出後のコード片からもわかる．

3.6 同値類分割

同値関係に基づき同値類分割を行う．同値類分割結果として抽出後のコード片とそれを記述した人数 (分類数) を示

Listing 3: power 関数の記述例 1

```
double power(double x,int n){
  int i;
  double ans;
  ans=1;
  for (i=1;i<=n;i++) {
    ans *= x;
  }
  return ans;
}
```

Listing 4: power 関数の記述例 2

```
double power(double x,int n){
  double ans=1;
  int i;
  for (i=0;i<n;i++) {
    ans = ans * x;
  }
  return ans;
}
```

Listing 5: power 関数の記述例 3

```
double power(double x, int n){
  double p;
  int i;
  for (i=0;i<n;i++) {
    p = p * x;
  }
  return p;
}
```

Listing 6: power 関数の記述例 4

```
double power(double x, int n){
  int a, i;
  a=1;
  for (i=0;i<n;i++) {
    a = a * x;
  }
  return a;
}
```

Listing 7: power 関数の記述例 (制御文の抽出後)

```
for{
}
```

Listing 8: power 関数の記述例 1(すべての抽出後)

```
double=1
for{
  double=double*double
}
return double
```

Listing 9: power 関数の記述例 2(すべての抽出後)

```
double=1
for{
  double=double*double
}
return double
```

Listing 10: power 関数の記述例 3(すべての抽出後)

```
for{
  double=double*double
}
return double
```

Listing 11: power 関数の記述例 4(すべての抽出後)

```
int=1
for{
  int=int*double
}
return int
```

すことで教員のコーディング状況把握を支援する．Listing 8，9，10，11 の同値類分割結果を図 1 に示す．図左が制御文のビュー，図右が演算子と型のビューである．それぞれのビューの表の左が抽出したコード片であり，右が分類数である．

制御文のビュー

for{ }	4

演算子と型のビュー

double=1
for{
double=double*double
}
return double
	2

for{
double=double*double
}
return double
	1

int=1
for{
int=int*double
}
return int
	1

図 1: 同値類分割例

3.7 設計と実現

TEBA[8] の構文解析機能を用いて 3 章で述べた補完，正規化，抽出の各処理を実現した．型の抽出機能では変数宣言時の変数の型を調べることにより変数名を型名へ置き換えている．

4. 評価

4.1 概要

C 言語を一通り学んだ学部 3 年生 11 人に協力してもらい，60 分の演習形式で**図 2** の関数を作成する問題を実行例とともに出題した．出題した問題では条件分岐，繰返し，配列，ポインタ，再帰関数などの C 言語学習の主要な単元を網羅している．演習後，1 分毎に取得した各学生のソースコードを対象に提案した同値類分割を行い，次の観点からその有効性を確認する．

観点 1　教員が確認すべきソースコードの数を減らすことができるか

観点 2　それぞれのビューを指導の判断材料とできるか

観点 1 は，制御文のビュー，演算子と型のビューの同値類分割の分割数により判断する．制御文のビューについて，問題 1，3，4，5 の分割数は少ないと予想される (そもそも問題 3 は制御文は不要である)．問題 2 は場合分けの方法が複数あるので，分割数は多いと予想される．演算子と型のビューについては，分割数は多いことが予想される．

観点 2 は，各ビューを教員が見てどのような指導が行えるか，また，その指導が実際のソースコードに対して適切かを判断する．制御文のビューでは処理の流れがわからない学習者に対する指導，演算子と型のビューでは型の誤りなどに対する指導が行えると予想される．

問題 1　整数 x の n 乗 (n>0) を返す関数
`int power(int x, int n)`

問題 2　実数 x の整数 n 乗を返す関数
`double power(double x, int n)`

問題 3　秒を分，秒の形に変換してポインタ引数に格納する関数
`void time(int s, int *min, int *sec)`

問題 4　ユークリッドの互除法により最大公約数を求める再帰関数
`int euclid(int a,int b)`
以下にユークリッドの互除法のアルゴリズムを示す．

```
euclid(a, b) =  a   if b=0
                euclid(b,a mod b) otherwise
```

`a mod b` は `a` を `b` で割った余りを表す

問題 5　実数を交換する関数とそれを用いて要素数 n の実数配列を逆順にする関数
`void swap(double *x, double *y)`
`void reverse(double a[], int n)`

図 2: 演習で出題した問題

各問題について，main 関数はあらかじめ作成済みである．学生からの質問には原則回答しないが，評価で用いた WebIDE の使用方法や数学的な知識に関する質問は例外として回答した．

4.2 結果

今回は同値類分割の効果を確認したいので，学生が各問題を開始した時刻からの経過時間を基に同値類分割を行う．学生によって各問題を終えるまでの時間が異なるので，演習開始からの時間を基にすると，後半になるにつれて解答中の問題が分散し，効果が十分に確認できない．なお，学生が問題の解答を終えた後も同値類分割対象のソースコードとして含めている．同値類分割に要した時間は約 4.8 秒であった (CPU: Intel Core i5 3.1GHz, メモリ: 8GB)．

4.2.1 観点 1

各問題の 2 分，5 分，10 分，15 分，30 分時点のデータについて分割数を示し，評価を行う．

制御文のビューによる分割数を表 2 に示す．事前に予想したように，問題 1，3，5 では分割数は少なく，問題 2 では多くなった．問題 4 では再帰関数を適切に記述できている学生が少ないことから記述が分散し，予想に反し分割数が多くなった．

演算子と型のビューによる分割数を表 2 に示す．ほとんどの問題で分割数は多いが，問題 1 では 5 まで減らせている (問題 1 のビューの推移は付録に示す)．問題 5 の swap 関数も分割数は少ない．swap 関数の正しい記述の同値類の分類数は 2 分時点で 6，5 分時点で 8，10 分時点で 9，15 分時点で 10 である．比較的単純な問題では演算子と型のビューでも分割数が減ることがわかった．

4.2.2 観点 2 制御文のビュー

図 3 左は問題 1 の 15 分時点の制御文のビューである．while と for の二重ループの学生がいることがわかり，こ

表 1: 制御文のビューによる分割数

分割数/時間	2 分	5 分	10 分	15 分	30 分
問題 1	2	4	4	4	-
問題 2	4	7	8	7	-
問題 3	3	5	5	4	-
問題 4	4	6	6	6	5
問題 5(swap)	2	1	1	1	1
問題 5(reverse)	2	2	3	2	2

表 2: 演算子と型のビューによる分割数

分割数/時間	2 分	5 分	10 分	15 分	30 分
問題 1	10	9	5	5	-
問題 2	8	10	11	10	-
問題 3	11	9	8	8	-
問題 4	4	10	11	11	11
問題 5(swap)	5	4	3	2	2
問題 5(reverse)	6	8	8	7	8

の問題では繰返し 1 回でよいとの指導が有効である．if による条件分岐を記述している学生もいるが，これは n が 0 かどうかで場合分けしていると考えられる．実際のソースコードを確認したところ，if を記述していた学生は予想通り n が 0 であるか確認していた．問題 1 ではこのような場合分けは不要ではあるが，特に指導は必要ない．

図 3 中は問題 3 の 5 分時点の制御文のビューである．1 行目の 6 人は制御文を書いていない学生である．この問題では，整数の除算 (/) と剰余算 (%) を用いるだけでよいが，if による条件分岐や while による繰返しを記述している学生がいる．これまでの経験より，C 言語における剰余算 (%) がわからず，繰返し引くことで余りを求める学生がいることがわかっている．while の学生には剰余算について指導するとよい．実際のソースコードを確認したところ，予想通り繰り返し減算を行っていた．for を用いている学生にも同様の指導が有効である．

図 3 右は問題 4 の 10 分時点の制御文のビューである．if により場合分けして，再帰関数を呼び出して return すればよいが，while による繰返しや if の入れ子を記述している学生がいる．while を記述している学生には一つの if-else 文で記述できると指導するとよい．実際のソースコードを確認したところ，if の入れ子の学生はユークリッドの互除法を適用する a,b の大小関係を比較していた．while を用いていた学生は 3 人でありその内容は，while 内で再帰関数を呼び出している，while 文を用いて実装している，while 以外の記述はない，であった．これらの学生に先の指導では不十分であり，追加の指導が必要になると考えられる．

制御文のビューで処理の流れがわかっていない学生に対して指導が行えることが確認できた．

4.2.3 観点 2 演算子と型のビュー

図 4 左は問題 1 の 15 分時点の演算子と型のビューである．制御文のビューでは for を適切に書いていると判断した学生の 1 人 (4 行目) は再帰呼出しで計算しようとしていたことがわかる．再帰呼出しでも累乗は計算できるが，for は不要であるので，そのことを指導するとよい．

問題 1: 15 分時点

for{ }	8
while{ }	1
while{ for{ } }	1
if{ } else{ for{ } }	1

問題 3: 5 分時点

	6
if{ }	2
for{ }	1
if	1
while{ }	1

問題 4: 10 分時点

if{ } else{ }	3
if{ }	3
while{ }	2
	1
while	1
if{ if{ } else{ } } else{ } if{ } else{ }	1

図 3: 結果: 制御文のビュー

図 4 右は問題 3 の 5 分時点の演算子と型のビューである．制御文のビューでは制御文を書いていなかった 2 人は，代入文も書いていなかったことがわかる (2 行目)．ポインタは適切に利用できている学生が多いが，関節参照演算子 * を書けていない学生 (4 行目) もおり，指導の対象となる．

図 5 は問題 4 の 10 分時点の演算子と型のビューである．同値類分割がまとまることはなかったが，再帰呼出しで return をしていない学生が複数いることがわかる．制御文のビューでは if を書けている学生に指導の必要はないとしたが，if を書けていても再帰関数を書けていない学生が図左の 3 行目，右の 2，3，4 行目の 4 人いることがわかる．if の入れ子の学生も再帰関数呼び出しを return していない．while の学生について，図左の 6 行目の学生には再帰関数では while が不要であること，図右 1 行目の学生には再帰関数を用いて作成するように指導することができる．

紙面の都合で結果を載せられないが，問題 5 の reverse 関数の 30 分時点での演算子と型のビューでは，模範解答と同じ for{ swap(&double[int],&double[int-int-1])} の同値類の分類数が 4 で，残りは分類数 1 の同値類が 7 つある．その中の swap 関数の呼出しには，swap(*double[int],*double[int]) や swap(double[int],double[int-int-1]) などのポインタの誤りがあり，指導の対象となる．

4.3 評価

本研究では制御文や演算子，変数の型などを抽出しているが，これらの情報だけでも明らかに誤っているプログラムは把握できる．もともとのソースコードに比べれば，少

問題 1: 15 分時点

int=1
for{
int=int*int
}
return int
	7

int=1
while{
int=int*int
int=int-1
}
return int
	1

int=int
if{
return 1
}
else{
for{
int=int*int
}
return int
}
	1

int=0
for{
int=int*
power(int,int)
}
return int
	1

while{
for{
return int
}
}
	1

問題 3: 5 分時点

*p_int=int/60 *p_int=int%60	2
//time()	2

*p_int=0
while{
int=int-60
*p_int=*p_int+1
}
*p_int=int
	1

if{
p_int=int/60
int=int-(p_int*60)
}
	1

if	1

for{
int=int+1
}
	1

*p_int=int/60 *p_int=int-(*p_int*60)	1

*p_int=int/60
*p_int=int%60
if{
}
	1

int=int *p_int=int/60 *p_int=int-(*p_int*60)	1

図 4: 結果: 演算子と型のビュー 1

問題 4: 10 分時点

	1
while	1

if{
int=int%int
}
return int
	1

if{
return euclid(int,int%int)
}
return int
	1

if{
int=int%int
if{
return int
}
else{
euclid(int,int)
}
}
else{
int=int%int
}
if{
return int
}
else{
euclid(int,int)
}
	1

while{
return euclid(int,int%int)
}
	1

int=int%int
while{
int=int
int=int
int=int%int
}
return int
	1

if{
return int
}
else{
int=int%int
}
euclid(int,int)
	1

if{ }	1

if{
euclid(int,int%int)
}
else{
}
	1

int=int
int=int%int
if{
return int
}
else{
euclid(int,int)
}
	1

図 5: 結果: 演算子と型のビュー 2

ない情報でコーディング状況を把握でき，提案方法は有効である．

分類数が多い同値類は正解であることが多いことがわかった．間違った記述についても一部の分類数が増えると予想したが，今回は多くの誤答は分類数が 1 であった．これは評価を行った人数が少ないことが関係していると考えられる．

5. おわりに

本研究ではプログラミング演習において学習者のコーディング状況の把握を行うために，制御文や代入文における演算子や変数の型などを用いて同値類分割する方法を提案した．実際に演習を行い，提案方法の有効性を確認した．

今後の課題として，より多くの学習者や問題を対象として評価を行うこと，教員へ同値類分割結果を提示するインタフェースの考察，模範解答などと比較した指導が必要な同値類の自動判定などが挙げられる．

謝辞

卒業研究として一緒に取り組んだ松原茂輝君と高畑弘昭和君，演習に協力してくれた蜂巣研究室の学生に感謝します．本研究の一部は，JSPS 科研費 17K00114，17K01154，2018 年度南山大学パッヘ奨励金 I-A-2 の助成を受けた．

参考文献

[1] 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(2013), pp.330-339.

[2] 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員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(2011), pp.3135-3149.

[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(2012), pp.67-72.

[4] 蟹江茉衣香, 松原知奈美, 佐竹玲己衣 :『プログラミング演習における制御構造と条件式を用いた進捗状況把握方法の提案』. 南山大学情報理工学部 2015 年度卒業論文, 2016.

[5] 蜂巣吉成, 吉田敦, 阿草清滋 :『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』. 情報処理学会研究報告, CE-125, NO.3, 2014.

[6] 長島和平, 堀越将之, 長慎也, 間辺広樹, 兼宗進, 並木美太郎 :『プログラミング学習支援環境 Bit Arrow の教員支援機能の設計と試作』. 情報教育シンポジウム論文集, 18 号 (2017), pp.129-136.

[7] Abdullah Sheneamer, Jugal Kalita: “A Survey of Software Clone Detection Techniques” , International Journal of Computer Applications, 137(10), pp.1-21, March 2016.

[8] 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(2012), pp.1832-1849.

[9] 柴田望洋 :『新・明解 C 言語 入門編』, SB Creative, 2014

付　　録　　問題 1 の制御文のビューと演算子と型のビューの推移

各時点の上が制御文のビュー，下が演算子と型のビューである．2 分時点の演算子と型のビューでは，^を使って累乗を計算しようとしている学生がいることがわかる．

2 分時点

for{	7
}	
	4

	2
for{ power(int,int-1) }	1
for{ int=int*int }	1
int=Undef return int	1
for{ Undef=Undef }	1
int=1 for{ int=int*int } return int	1
int=int^int return int	1
for{ int=int*int }	1
for{ }	1
int=1 for{ }	1

5 分時点

for{	8
}	
while{	1
}	
if{	
}	
else{ for{ }	
}	1
	1

int=1 for{ int=int*int } return int	3
for{ int=int*int } return int	1
int=0 for{ int=int*int } return int	1
for{ } int=int*int return int	1
int=int^int return int	1
for{ }	1
int=1 while{ int=int*int int=int-1 } return int	1
for{ int=int*power(int,int-1) } return int	1
int=int if{ return 1 } else{ for{ int=int*int } return int }	1

10 分時点

for{	8
}	
if{	
}	
else{ for{	
}	1
}	
if{	1
}	
while{	1
}	

int=1 for{ int=int*int } return int	7
int=int if{ return 1 } else{ for{ int=int*int } return int }	1
int=1 while{ int=int*int int=int-1 } return int	1
if{ return int*power(int,int-1) }	1
for{ int=int* } return int	1

15 分時点

for{	8
}	
while{	1
}	
while{ for{	
}	1
}	
if{	
}	
else{ for{	
}	1
}	

int=1 for{ int=int*int } return int	7
int=1 while{ int=int*int int=int-1 } return int	1
int=int if{ return 1 } else{ for{ int=int*int } return int }	1
int=0 for{ int=int* power(int,int) } return int	1
while{ for{ return int } }	1