N-029

プログラミングの実習における学生のプログラムの題意と異なる実装の傾向調査と一考察
Trend Survey and Consideration of Inappropriate Implementation of Students' Source Code in Programming Classes

杉田 基樹† 早川 智一‡
Motoki Sugita Tomokazu Hayakawa

1. はじめに

近年，大学などの教育機関では，プログラミングの実習（以下，実習）の運営を支援するシステム（以下，実習システム）を用いて学生の提出物（プログラム）を自動で評価している [1] [2]．これは，

1. 実習の課題数や履修者数に応じて提出物の数が増大するため，限られた時間で教員が提出物の評価を手作業で行うことが困難である [3]，
2. 提出物の評価を自動化することで，それ以外の業務（学生との密接なコミュニケーションや教材の高品質化など）に教員がより多くの工数を充てられる [4]

――ためである．本学科でも，1 年次の学生を対象に，専用の実習システム MAX/C [5] を用いた C 言語の実習「プログラム実習」を行っている．

学生の提出物の中には，実行結果は正しくても実装方法が題意（出題者の意図）と異なるものがある [6]．なぜならば，学生は，品質の高いプログラムよりも正しく動作するプログラムの作成に重点をおく傾向にある [7] [8] ためである．例えば，「反復処理を用いて整数を 1 から 5 まで 1 行ごとに出力するプログラムを C 言語で作成せよ」という課題では，`for` や `while` を用いて `printf` を 5 回呼び出す実装を意図している．一方で，`printf` を 1 回だけ呼び出し，文字列 `"1\n2\n3\n4\n5\n"` を出力する実装は，反復処理を用いていないため明らかに題意と異なる．

実習システムは，題意と異なる実装を受理する（正しいと評価する）べきではない．これは，実習システムが題意と異なる実装を受理すると，学生は自分の実装が題意と異なることに気づかないまま学習を進めてしまい，結果的に学生の誤った理解を助長しかねないためである．

しかし，実習システムは，題意と異なる実装の提出物を検出できずに受理する場合がある．なぜならば，実習システムの多くは提出物の評価手法に入出力テストを用いている [2] [6] が，入出力テストは，あらかじめ定義した入力に対する出力の正しさのみで正答かどうかを判定する手法であり，出力が正しければどのような実装でも正しいと判定するためである．例えば，入出力テストのみを用いた実習システムは，前述の課題の反復処理を用いていない実装も受理する可能性がある．

†明治大学大学院理工学研究科 Graduate School of Science and Technology, Meiji University
‡明治大学理工学部 School of Science and Technology, Meiji University

先行研究 [6] [9] では，実習システムでの題意と異なる実装の自動検出手法が提案されているが，実際の提出物の全数調査に基づくものは少なく，これらの手法で検出できない実装が存在する可能性があると我々は考える．

本研究の目的は，既存の手法での自動検出が困難な題意と異なる実装を見つけ出し，それに対する検出手法を提案することで，実習の質を向上させることにある．そこで我々は，本学科のプログラム実習で MAX/C が受理したソースコード 22,383 件を全数調査した．調査の結果，我々は 9 種類の題意と異なる実装のパターンを抽出した．本論文では，抽出したパターンごとの発生要因と，各パターンに対する検出方法を考察する．

本論文の構成は次のとおりである．2 章では関連研究を紹介する．3 章ではプログラム実習について概説する．4 章では題意と異なる実装のパターンの抽出方法を説明し，抽出したパターンを列挙する．5 章では各パターンの検出手法を考察する．6 章では今後の展望を述べる．

2. 関連研究

2.1 題意と異なる実装の自動検出

漆原ら [6] は，提出物が題意通りかを抽象構文木（以下，AST）で確認する手法を提案している．漆原らの提案手法では，提出物のソースコードを AST に変換し，AST が含むべき部分木を教員が課題ごとに指定可能にすることで，提出物が題意通りの構造を持つかを判定している．

横井ら [9] は，関数ごとに入出力テストが可能な C 言語のインタプリタを提案している．横井らの提案手法では，入出力テストの対象とする関数名と入出力を指定可能にすることで，プログラム全体だけでなく関数ごとの実装が題意通りかを判定している．

これらの研究では，実習システムによる題意と異なる実装の自動検出を可能にしているが，本研究では，これらの既存の手法で検出が困難な題意と異なる実装の調査と，その検出手法を提案する．

2.2 学生の提出物の誤りの分析

Albrecht ら [10] は，C 言語のプログラミング入門コースでの学生の提出物を分析し，学生が犯しやすいエラーを 6 つのカテゴリに分類している．学生が特に犯しやすいエラーとして出力形式の誤りや入力値のバリデーション不足などを挙げ，これらの主な発生原因が，学生のコーディングの「ずさんさ（Sloppiness）」にあると指摘している．また，エラーのカテゴリの中には，課題に対する「誤

解（Misinterpretation）」が含まれており，学生の提出物から発見したエラーの中には，題意の誤解によるエラーが存在すると述べている．

Albrecht らの研究は，学生の提出物を分析し，学生の提出物の誤りの傾向を調査している点で本研究と類似している．一方で，Albrecht らが課題に不正解であった（エラーがあった）提出物を分析しているのに対し，本研究では課題に正解した提出物を分析し，エラーではないが題意とは異なる実装の傾向を調査している点で異なる．

3. プログラム実習

プログラム実習は，本学科の 1 年次を対象とした C 言語の実習である．プログラムを作成する課題が 1 回の実習につき 6〜10 題出題され，1 年間の全 28 回の実習で合計 178 題が出題される．各回の課題は，レベル 1（低難易度）〜レベル 5（高難易度）までに分けられる．履修する学生はプログラミング初学者を想定しており，特に初めのうちはプログラミングの基礎的な知識を身に着けるための課題が出題される．

実習の運営には専用の実習システム MAX/C を用いている．MAX/C は，学生への課題の出題と学生の提出物の評価を担う．学生は，課題の指示にしたがって C 言語のプログラムを作成し，MAX/C に提出して評価を受ける．MAX/C に正しいと評価された場合は完了となり，正しくないと評価された場合は再提出する必要がある．

4. 題意と異なる実装のパターンの抽出

我々は，学生の提出物の題意と異なる実装にどのようなパターンがあるかを明らかにするため，以下の 3 段階に分けて学生の提出物を分析した．

第 1 に，MAX/C の全 178 題を次の流れで調査し，題意と異なる実装のパターン一覧を作成した：(1) 課題文を精読し題意を把握する；(2) 受理された提出物の実装を人力で確認する；(3) 発見したパターンを一覧に追加する．

第 2 に，各パターンの重大度を次の 3 つに分類した：

重大度（大） 題意と明らかに異なり，正しいと評価すると学生の学びを大きく阻害する恐れがある；

重大度（中） 題意と完全に異なるとは言い切れないが，正しいと評価すると学生の学びを少なからず阻害する恐れがある；

重大度（小） 題意と完全に異なるとは言い切れず，初学者であることを鑑みるとやむを得ない．

第 3 に，各パターンの発生件数を調査した．パターンの検出には，ソースコード評価ツールによる検出，構文解析による検出，目視での検出などを併用した．

我々は，9 種類の題意と異なる実装のパターンを抽出した．以下に，各パターンの特徴・ソースコード例・発生理由・重大度を示す．なお，ソースコード例は実際の提出物に基づくが，紙面の都合上，不要なスペースや改行を削除した上でコード整形を施してある．

4.1 課題の指示に従わないパターン

実習システムの課題の一部では，学習させる内容や実習の進行の都合上，プログラムの実装方法（使用すべき文法やアルゴリズムなど）に指示がある．そのような課題で，指示に従わない実装が見受けられた．

実装方法に指示がある課題の例として，「9 の i 乗を i = 0 から i = 5 まで示す表を作成するプログラムを実装せよ」という課題（以下，「べき乗の表」）を挙げる．「べき乗の表」では，処理の手順が以下のように指示されている：(1) 変数 p に 1 を代入する；(2) `"9^0 = "` に続けて p の値を出力し改行する；(3) 変数 p の値を 9 倍する；(4) `"9^1 = "` に続けて p の値を出力し改行する；(5) 変数 p の値を 9 倍する；(6) `"9^2 = "` に続けて p の値を出力し改行する；(7) 9 の 5 乗の出力を行うまで繰り返す．ここでは，変数 p の値を変えながら出力するよう指示がある．「べき乗の表」の指示通りの例を**コード 1** に示す．

これに対し，「べき乗の表」の指示に従わない例を**コード 2** に示す．この実装の出力はコード 1 と同じであるため，実習システムには受理された．しかし，変数 p の値は変化しておらず，この実装は明らかに課題の指示に従っていない．したがって，我々はこのパターンを重大度（大）に分類した．

ここでは，課題の指示に従わないパターンの 1 つとして，処理の手順が指示通りでないパターンを挙げたが，この他にも，課題の指示に従わないパターンに属するものがあることがわかった．それらを以下に示す．

4.1.1 使用する制御文が指示通りでないパターン

課題で指定された制御文を使用していない実装が見受けられた．特に，`for` を指定されているのに `while` を使用している実装（及びその逆）が多く見受けられた．

4.1.2 プログラムの構造が指示通りでないパターン

制御文と同様に，指定されたプログラムの構造でない実装が見受けられた．特に，再帰呼び出しの使用を指示されているのに使用していない実装が多く見受けられた．

コード 1 「べき乗の表」の指示通りの例

```c
#include <stdio.h>
int main(void)
{
    int p = 1;
    printf("9^0 = %d\n", p);
    p = p * 9;
    printf("9^1 = %d\n", p);
    p = p * 9;
    printf("9^2 = %d\n", p);
    p = p * 9;
    printf("9^3 = %d\n", p);
    p = p * 9;
    printf("9^4 = %d\n", p);
    p = p * 9;
    printf("9^5 = %d\n", p);
    return 0;
}
```

コード 2　「べき乗の表」の指示に従わない例

```c
#include <stdio.h>
int main(void)
{
    int p = 1;
    printf("9^0 = %d\n", p);
    printf("9^1 = %d\n", p * 9);
    printf("9^2 = %d\n", p * 9 * 9);
    printf("9^3 = %d\n", p * 9 * 9 * 9);
    printf("9^4 = %d\n", p * 9 * 9 * 9 * 9);
    printf("9^5 = %d\n", p * 9 * 9 * 9 * 9 * 9);
    return 0;
}
```

4.1.3　マクロ置換を使用していないパターン

マクロ置換の使用を指示されているのに使用していない実装が見受けられた．また，#define で定義したマクロを 1 度も使っていない実装も存在した．

4.2　出力をハードコードしているパターン

一部の課題には，入力が存在せず出力が固定になるものがある．そのような課題で，出力をハードコードしている実装が見受けられた．

出力が固定になる課題の例として，4.1 節の「べき乗の表」を挙げる．「べき乗の表」の出力をハードコードしている例をコード 3 に示す．この実装では，プログラムが計算するはずの 9 のべき乗を学生が計算してハードコードしており，明らかに題意と異なる．したがって，我々はこのパターンを重大度（大）に分類した．

4.3　前の課題の関数を使用していないパターン

一部の課題では，前の課題で作成した関数を使用する必要がある．そのような課題で，必要な関数を使用していない実装が見受けられた．

前の課題で作成した関数を使用する課題の例として，「n の階乗を求める関数 `int fact(int n)` を作成せよ」という課題の後の，「n 個の集合から要素を k 個取り出すときの組み合わせ数を求める関数 `int combi(int n, int k)` を，前問の `fact` を用いて作成せよ」という課題（以下，「組み合わせ数」）を挙げる．「組み合わせ数」で前の課題の関数を使用している例をコード 4 に示す．

これに対し，前の課題の関数を使用していない例をコード 5 に示す．この実装では，階乗を求める関数 `fact` が無く，関数 `combi` の中に階乗を求める処理が含まれているため，明らかに題意と異なる．したがって，我々はこのパターンを重大度（大）に分類した．

コード 3　「べき乗の表」の出力をハードコードしている例

```c
#include <stdio.h>
int main(void)
{
    int p = 1;
    printf("9^0 = %d\n", p);
    printf("9^1 = %d\n", p * 9);
    printf("9^2 = %d\n", p * 81);
    printf("9^3 = %d\n", p * 729);
    printf("9^4 = %d\n", p * 6561);
    printf("9^5 = %d\n", p * 59049);
    return 0;
}
```

コード 4　「組み合わせ数」で前の課題の関数を使用している例

```c
int fact(int n)
{
    int i, value = 1;
    for (i = 1; i <= n; i++)
        value *= i;
    return value;
}

int combi(int n, int k)
{
    return fact(n) / (fact(k) * fact(n - k));
}
```

コード 5　「組み合わせ数」で前の課題の関数を使用していない例

```c
int combi(int x, int y)
{
    int p = 1, q = 1, r = 1, i;
    for (i = 1; i <= x; i++)
        p = p * i;
    for (i = 1; i <= y; i++)
        q = q * i;
    for (i = 1; i <= x - y; i++)
        r = r * i;
    return p / (q * r);
}
```

4.4　関数の役割分担が適切でないパターン

関数ごとの役割（仕様）が細かく定義されている課題で，関数の役割分担が適切でない実装が見受けられた．関数の役割が細かく定義されている課題の例として，「端末から次々と入力される整数の総和を求めるプログラムを作成せよ」という課題（以下，「入力整数の総和」）を挙げる．「入力整数の総和」の課題文には，さらに以下の情報が含まれる：(1) 最初に整数の個数 n が入力され，その後に n 個の整数が入力される；(2) 関数 `main` で n 個の整数を配列に格納する；(3) 配列 a の最初の n 要素の和を返す関数 `int sum(int a[], int n)` を用いる；(4) n の値は 1,000 以下と仮定してよい．ここでは，関数 `main` が整数の入出力，関数 `sum` が総和の計算の役割を担うことを想定している．「入力整数の総和」の関数の役割が適切な例をコード 6 に示す．

これに対し，関数の役割分担が適切でない例をコード 7 に示す．この実装では，関数 `main` が担うべき役割（整数の入力の受け取りや結果の表示）を関数 `sum` が担っているため，明らかに題意と異なる．したがって，我々はこのパターンを重大度（大）に分類した．

4.5　配列の要素数が必要数より 1 多いパターン

配列を用いる課題で，配列の要素数が必要数より 1 多い実装が見受けられた．配列を用いる課題の例として，「標準入力から整数を 10 個受け取り，逆順で出力するプログラムを実装せよ」という課題（以下，「逆順の出力」）を挙げる．「逆順の出力」で配列の要素数が適切な例をコード 8 に示す．この課題では，入力される整数が 10 個であると決まっているため，要素数が 10 の配列を宣言し，`a[0]` から `a[9]` に値を格納することを想定している．

これに対し，配列の要素数が必要数より 1 多い例を

コード 6　「入力整数の総和」の関数の役割が適切な例

```c
#include <stdio.h>
int sum(int a[], int n)
{
    int i, sum = 0;
    for (i = 0; i < n; i++)
        sum += a[i];
    return sum;
}

int main(void)
{
    int i, n, a[1000];
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    printf("The sum of %d input numbers is %d.\n", n,
           sum(a, n));
    return 0;
}
```

コード 7　「入力整数の総和」の関数の役割が適切でない例

```c
#include <stdio.h>
int sum(int a[1000], int n)
{
    int i, s = 0;
    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
        s = s + a[i];
    }
    printf("The sum of %d input numbers is %d.\n", n, s);
    return 0;
}

int main(void)
{
    int x[1000], y;
    scanf("%d", &y);
    sum(x, y);
    return 0;
}
```

コード 9 に示す．この実装では，要素数が 11 の配列を宣言している．このような実装が見受けられた理由としては，学生が 0 から開始する添え字に慣れておらず，0〜9 の代わりに 1〜10 を使用したためと考えられる．

課題によっては，配列の要素数の必要数が題意からは特定できない場合があるため，題意と完全に異なるとは言い切れない．しかし，このような実装を正しいと評価すると，要素数を常に 1 多く宣言する癖を助長しかねないため，我々はこのパターンを重大度（中）に分類した．

4.6　特定の入力でしか動作しないパターン

課題で与えられた特定の入力でしか実行結果が正しくならない実装が見受けられた．

例えば，「階乗を計算する関数を用いて，1 から 10 までの整数の階乗を小さい順に出力するプログラムを作成せよ」という課題（以下，「階乗」）を挙げる．「階乗」では，

コード 8　「逆順の出力」で配列の要素数が適切な例

```c
#include <stdio.h>
int main(void)
{
    int i, a[10];
    for (i = 0; i < 10; i++)
        scanf("%d", &a[i]);
    for (i = 9; i >= 0; i--)
        printf("%d ", a[i]);
    return 0;
}
```

コード 9　「逆順の出力」で配列の要素数が必要数より 1 多い例

```c
#include <stdio.h>
int main(void)
{
    int i, a[11];
    for (i = 1; i < 11; i = i + 1)
        scanf("%d", &a[i]);
    for (i = 10; i > 0; i = i - 1)
        printf("%d\n", a[i]);
    return 0;
}
```

与えられた整数の階乗を計算する（任意の名前の）関数を定義し，1 から 10 までの整数に対してこの関数を関数 `main` から繰り返し呼び出す実装を想定している．

特定の入力でしか動作しない例をコード 10 に示す．この実装では，仮引数 `x` として与えられた整数の階乗を返す関数 `int f(int x)` を定義しているが，`x` が 1 から 10 の範囲にあると決めてしまい，それぞれの場合の結果を返すのみであるため，関数 `f` は階乗を計算する関数とは言えない．課題文に「1 から 10 まで」と明記されているため，必ずしも課題と異なるとは解釈できないが，このような実装を正しいと評価すると，実習システムに受理されることのみを目的としたコーディングを助長しかねないため，我々はこのパターンを重大度（中）に分類した．

4.7　繰り返しを適切に使用していないパターン

`for` や `while` などでの繰り返しを使用すべき場所でこれらを適切に使用していない実装が見受けられた．繰り返しを用いるべき課題の例として，4.5 節の「逆順の出力」を挙げる．「逆順の出力」で，繰り返しを適切に使用していない例をコード 11 に示す．この実装では，`for` を用いて整数の入力を受け取っているが，出力では `for` を用いず，配列の中身を 1 要素ずつ参照している．

この課題で扱う整数は 10 個だが，仮に 100 個に変更された場合，このような実装では修正が困難である．しかし，学生はプログラムの仕様変更を考慮する余裕がなく，プログラムの拡張性を考慮せずにコーディングする傾向にあると考えられる．プログラムの拡張性は高い方が望ましいが，拡張性の高さの基準は自明ではなく，必ずしも題意と異なるとは言い切れないため，我々はこのパターンを重大度（小）に分類した．

4.8　配列を適切に使用していないパターン

配列を使用すべき場所で，配列を適切に使用していない実装が見受けられた．具体的には，まとまった N 個のデータを扱う必要のある場面で，長さ N の配列ではなく N 個の変数を使用する実装が散見された．

この実装は，4.7 節と同様に，プログラムの拡張性を考慮していないことに起因すると考えられる．我々は，4.7 節と同様に，このパターンを重大度（小）に分類した．

4.9　関数 `main` での検証が不十分なパターン

一部の課題では，実装した関数を関数 `main` から呼び出し，動作の妥当性を学生自身に検証させる．そのような

コード 10　「階乗」で特定の入力でしか動作しない例

```c
#include <stdio.h>
int f(int x)
{
    if (x == 1)
        return (x);
    if (x == 2)
        return (1 * x);
    if (x == 3)
        return (1 * 2 * x);
    if (x == 4)
        return (1 * 2 * 3 * x);
    if (x == 5)
        return (1 * 2 * 3 * 4 * x);
    if (x == 6)
        return (1 * 2 * 3 * 4 * 5 * x);
    if (x == 7)
        return (1 * 2 * 3 * 4 * 5 * 6 * x);
    if (x == 8)
        return (1 * 2 * 3 * 4 * 5 * 6 * 7 * x);
    if (x == 9)
        return (1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * x);
    if (x == 10)
        return (1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * x);
}

int main(void)
{
    int n;
    for (n = 1; n <= 10; n++)
        printf("%d! = %d\n", n, f(n));
    return 0;
}
```

コード 11　「逆順の出力」で繰り返しを適切に使用していない例

```c
#include <stdio.h>
int main(void)
{
    int x, a[10];
    for (x = 0; x < 10; x = x + 1)
        scanf("%d", &a[x]);
    printf
        ("%d %d %d %d %d %d %d %d %d %d\n",
         a[9], a[8], a[7], a[6], a[5],
         a[4], a[3], a[2], a[1], a[0]);
    return 0;
}
```

課題では，実習システムでの提出物の評価の際に，関数 main をあらかじめ用意したものに差し替えて関数の動作を検証するが，それを逆手に取り，提出物の関数 main での学生自身による検証が不十分な実装が見受けられた．

例えば，「実装した関数を関数 main から 3 回以上呼び出し，それぞれ異なる値で検証せよ」という指示があるのに対し，1 回しか呼び出していない実装・1 回も呼び出さずに return 0; のみの実装・そもそも関数 main が存在しない実装などが散見された．

4.10　抽出件数

各パターンの検出件数を表 1 に示す．各パターンは集計対象とすべき提出物の総数が異なるため，検出割合は，集計対象とすべき提出物の総数を分母として算出した．

検出数としては 4.1 節が最も多かった．したがって，課題文で実装方法を細かく指示しても，実習システムで検出できなければ指示に従わずに提出する学生が多い傾向にあると言える．また，4.3 節，4.4 節，4.9 節の検出割合が比較的高いことから，関数に関係がある課題に誤りが多い傾向にあると言える．

表 1　各パターンの検出件数と割合

節	対象数	検出数	割合	対象課題
4.1	8,305	496	5.97%	実装に指示あり
4.2	4,413	18	0.41%	出力が固定
4.3	2,013	236	11.72%	前の課題の関数を使用
4.4	11,339	245	2.16%	関数を使用
4.5	10,044	192	0.86%	配列を使用
4.6	22,383	14	0.06%	全て
4.7	19,149	211	1.10%	繰り返しを学習済み
4.8	16,636	171	1.03%	配列を学習済み
4.9	7,158	433	6.05%	main を差し替え評価

5.　検出手法の考察

我々は，4 章で抽出した各パターンが既存の手法で自動検出可能かを調査した．調査対象の手法を表 2 に示す．A〜D は，C 言語のコンパイラ・静的解析ツール・単体テストフレームワークのうち，現在も更新が継続しているものから選定した．E と F は，論文の内容をもとに自動検出可能かを判断した．

調査結果を表 3 に示す．行が手法，列が各パターンを表す．記号の意味は以下のとおりである：○ はほとんどの実装を検出可能；△ は一部の実装を検出不可能；× はほとんどの実装を検出不可能．以下に，各手法で検出可能なパターンと困難なパターンの詳細を述べる．

5.1　既存の手法で自動検出が可能なパターン

コンパイラでは，4.9 節の一部が検出可能であった．具体的には，関数 main が存在しない場合や，文法にミスがある実装は検出できたが，return 0; のみ記述している実装や，他の関数の呼び出しが指定された回数に満たない実装は検出できなかった．

静的解析ツールでは，4.6 節の一部が検出可能であった．例えば，コード 10 に手法 C を適用すると，return が欠落したパスが検出された旨の警告が表示される．これは，関数 f の引数が 1〜10 のどれでもない場合の戻り値が考慮されていないためである．しかし，このパターンを本質的には検出できていない．例えば，関数 f の最後に return 0; などと記述すると警告が表示されなくなるが，特定の入力以外で階乗を求められないという本質的な誤りは解決できていない．

表 2　自動検出が可能かの調査対象の手法

記号	手法	Ver.	備考
A	GCC	13.1.0	コンパイラ
B	Clang	19.0.0	コンパイラ・静的解析ツール
C	Cppcheck	2.7	静的解析ツール
D	Criterion	2.4.2	単体テストフレームワーク
E	漆原ら [6]	なし	AST を用いた判定
F	横井ら [9]	なし	関数ごとのテスト

表 3　既存の手法で自動検出が可能かの調査結果

パターン	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9
重大度	大	大	大	大	中	中	小	小	大
A	×	×	×	×	×	×	×	×	△
B	×	×	×	×	×	×	×	×	△
C	×	×	×	×	×	△	×	×	△
D	×	×	△	○	×	○	×	×	△
E	△	△	○	△	×	△	×	×	○
F	×	×	△	○	×	○	×	×	△
全て併用	△	△	○	○	×	○	×	×	○

単体テストフレームワークでは，関数単体の入出力テストが可能なため，4.4 節をほとんど検出できた．4.6 節は，題意に沿う範囲でテストケースの網羅率を高めることでほとんどの実装を検出できた．4.3 節は，前の課題の関数が実装されていないものは検出できたが，実装された上で呼び出されていないものは検出できなかった．

漆原ら [6] の手法は，4.3 節・4.9 節 の実装をほとんど検出可能であると考えられる．なぜならば，これらのパターンは，AST が含むべき部分木（プログラムが持つべき構造）が題意から明らかなためである．一方で，4.1 節・4.2 節・4.4 節・4.6 節 は，部分木の定義方法によって検出が可能な実装と不可能な実装があると考えられる．これは，AST が含むべき部分木が題意から明らかでない場合があり，指定する部分木が単純すぎると題意と異なる実装を検出できず，複雑すぎると題意通りの実装を誤って検出する場合があるためである．

横井ら [9] の手法は，単体テストフレームワークと同様の結果になると考えられる．ただし，横井らの手法はインタプリタとして提案されているため，入出力テストでは検出が困難な誤りの検出が可能であると考えられる．

5.2　既存の手法で自動検出が困難なパターン

4.1 節は，漆原らの手法でのみ一部の実装が検出可能であった．しかし，変数の値を指示通りに変化させる課題では，特定の構造の有無のみでは検出できない場合がある．したがって，このパターンの検出には，実行時の変数の値を追跡する手法などが必要であると考えられる．

4.2 節は，漆原らの手法でのみ一部の実装が検出可能であった．このパターンが題意と異なる根本的な原因は，特定の構造を持たないことではなく，プログラムが算出すべき値を学生が算出していることにある．したがって，このパターンの検出には，プログラムが何を算出すべきかを教員が指定し，プログラムの実行時に計算過程を追跡する手法などが必要であると考えられる．

4.5 節・4.7 節・4.8 節は，調査対象のどの手法でも検出がほとんど不可能であった．これらを検出する別の方法として，（1）4.5 節は，ソースコード上で配列の要素数を確認する手法や，実行時に配列の 0 要素目（配列 `a` での `a[0]`）が使用されているかを確認する手法，（2）4.7 節は，似た記述が何度も現れるのを検出する手法，（3）4.8 節は，宣言可能な変数の数を制限し，それを超えて宣言していないかを確認する手法――などが考えられる．

6.　おわりに

本論文では，プログラミングの実習における学生の提出物の題意と異なる実装の傾向を明らかにするため，本学科の実習システムが受理した提出物を全数調査し，9 種類の題意と異なる実装のパターンを抽出した．また，実習システムによる自動検出によって実習の質を向上させるため，既存の手法での自動検出が困難なパターンを調査した．調査の結果，4 パターンは既存の手法でほとんど検出できたが，2 パターンは一部のみ検出でき，3 パターンはほとんど検出できないという結論を得た．

今後の展望としては，重大度が大きく既存の手法で完全には検出できないパターン（4.1 節や 4.2 節など）に対し，自動で検出可能な手法の提案を行う予定である．

参考文献

[1] 竹山祐太郎，横井翔太，杉田基樹，早川智一 : 実習の要件の差異の吸収に着目したプログラミングの実習システムの提案と試作，第 22 回情報科学技術フォーラム（FIT2023）講演論文集，pp. 325–328 (2023).

[2] J. L. Fernandez Aleman: Automated Assessment in a Programming Tools Course, *IEEE Transactions on Education*, Vol. 54, No. 4, pp. 576–581 (2011).

[3] Silva, D. B., Carvalho, D. R. and Silla, C. N.: A Clustering-Based Computational Model to Group Students With Similar Programming Skills From Automatic Source Code Analysis Using Novel Features, *IEEE Transactions on Learning Technologies*, Vol. 17, pp. 428–444 (2024).

[4] Skalka, J. and Drlík, M.: Development of Automatic Source Code Evaluation Tests Using Grey-Box Methods: A Programming Education Case Study, *IEEE Access*, Vol. 11, pp. 106772–106792 (2023).

[5] Fujii, S., Ohkubo, K. and Tamaki, H.: MAX/C on Sakai - A Web-based C-Programming Course, *International Conference on Computer Supported Education* (2018).

[6] 漆原宏丞，本多佑希，岸本有生，兼宗進 : 抽象構文木を利用したプログラミング理解度採点の試み，研究報告教育学習支援情報システム（CLE），Vol. 2021-CLE-35, No. 16, pp. 1–6 (2021).

[7] Izu, C. and Mirolo, C.: Exploring CS1 Student's Notions of Code Quality, *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, pp. 12–18 (2023).

[8] Keuning, H., Heeren, B. and Jeuring, J.: Code quality issues in student programs, *Proceedings of the 22nd Annual Conference on Innovation and Technology in Computer Science Education*, pp. 110–115 (2017).

[9] 横井翔太，早川智一 : C 言語のプログラムを関数ごとにテスト可能なウェブブラウザ上で動作するインタプリタの提案と試作，第 84 回全国大会講演論文集，Vol. 2022, No. 1, pp. 323–324 (2022).

[10] Albrecht, E. and Grabowski, J.: Sometimes It's Just Sloppiness - Studying Students' Programming Errors and Misconceptions, *SIGCSE '20: Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pp. 340–345 (2020).