プログラムの自動テストを応用した
プログラミング演習支援方式に
関する研究

平成18年度

三重大学大学院工学研究科
博士前期課程 電気電子工学専攻

小坂祐章

修士論文

プログラムの自動テストを応用したプログラミング演習支援方式に関する研究

平成１８年度修了

三重大学大学院工学研究科

博士前期課程　電気電子工学専攻

小坂　祐章

目次

第1章　はじめに

近年，あらゆる製品に情報技術が用いられるようになった．そのため，将来，技術者となる者はプログラミング能力を習得しておくことが望ましい．このようなことから，大学教育におけるプログラミング教育の重要性が高まっている．

理想的なプログラミング演習の流れを以下に示す．

(1) 講師が学習者に課題を与える．
(2) 学習者は課題で指示されたプログラムを書き，エラーのないことを確認したのちに，それを講師に提出する．
(3) 講師は学習者が作成したプログラムをチェックし，動作エラーなどの修正すべき点があれば，それをコメントとして学習者に返す．
(4) 学習者は講師の指摘に従ってプログラムを修正する．
(5) 修正すべき点がなくなるまで工程の(3)，(4)を繰り返す．

現状では，プログラムの作成に慣れていないため，十分なテストをせずに提出する学習者も多い．また，受講者が多人数となると，講師が演習時間中に学習者の作成した個々のプログラムに対してきめ細やかなコメントを返すことは難しい．修正すべき点を具体的に指摘されないため，プログラムに動作エラーがあっても，それに気付かない学習者がいる．プログラミングの初心者の場合には，特にこれが顕著になる．

以上のことから，2章で示すようなシステムを用いた研究が行われている．しかし，このような方法であれば，システムが学習者のプログラムに存在する動作エラーを指摘できなかったり，システムに適用し易いように課題を対応させるため学習者が課題を理解することができなかったりと，問題点が存在する．

本研究では，プログラミング演習では，エラーを修正する過程を通じて，どのようにプログラムを作成したらよいのかを自分で考えながら習得していくことが重要であり，自分の作成したプログラムのエラーに気付かなければ，プログラム作成の能力を高めることができないと考える．

そこで，プログラムの提出直後に動作エラーに気付くことのできる情報を学習者に提供するプログラミング演習支援方式を提案する．提案する演習支援方式を用いることで，プログラミングの初心者であっても，自分のプログラムの動作エラーに気付くことができ，エラーを修正していく中でプログラム作成の能力を高めることができるようになる．

本論文の構成を示す．第2章で，プログラミング演習を支援する方式についていくつか先行研究を紹介する．第3章で，本研究でプログラミング演習の支援に応用するソフトウェア開発におけるプログラムの自動テストについて述べ

る．第４章で，プログラムの自動テストをそのままの形でプログラミング演習に適用できるかを考察する．第５章で，プログラムの自動テストを応用したプログラミング演習支援方式を提案する．第６章では，提案する方式の有効性を確認するための実験の結果を述べる．最後に，第７章でまとめる．

第2章　先行研究

プログラミング演習において，講師の負担を軽減するために，さまざまな研究が行われている．2章では，学習者が作成したプログラムから，誤りを自動的に検出する研究を紹介する．

2.1　出力形式を指定した課題を用いた方法

2.1はソフトウェア開発で行われているプログラムの自動テストをプログラミング演習に適用する手法である [1]．プログラムの入力と出力を厳密に定めておき，あらかじめ用意しておいたいくつかのテストケースを用いてプログラムの動作確認を自動的に行う．

学習者が初心者である場合，厳密に出力の形式が指定されたプログラムを作成することは難しい．文献 [1] では，課題の意図に対して間違いではないが，出力の形式を満たせない学習者が半数以上いることが報告されている．

2.2　標準プログラムとの比較

課題に対する標準的なプログラム，または評価済みのプログラムをいくつか用意しておき，それらのプログラムと学習者の提出したプログラムとの類似性を調べることで，プログラムを自動的に評価する [2]．

課題を満たすプログラムは無数に存在する．標準的なプログラムとの類似度を見るだけでは，これらの全てに対応することが難しい．文献 [2] では，課題を満たすプログラムのうち，ほぼ半数のプログラムだけが課題を満たしていると判定できたことが報告されている．

2.3　典型的な間違いの検出

初心者がおかしがちな間違いとその検出方法をあらかじめ用意しておき，それらを用いてプログラムの誤りを自動的に検出する [3]．

課題に応じた間違いを検出できるようにするためには，その検出方法をシステムに組み込む必要がある．講師が課題を提供する度にこれを行うことは難しい．

2.4　出力に関する条件を緩めたプログラムの自動テスト

2.1 で述べたように，プログラムの入力と出力を厳密に定めておいて，初心者の作成したプログラムに自動テストを行うことは難しい．そこで，望月らは，文献 [4] において，出力に関する条件を緩めたプログラムのテスト方法を提案

した．この方法では，プログラムの出力に必ず現れるはずの文字列が含まれているかどうかのみを確認する．これにより，出力の形式が厳密に定められていない課題の場合，あるいは，学習者が出力の形式を厳密に守れない場合に対しても，プログラムの自動テストが行えるようになる．

この方法では，出力に必ず現れるはずの文字列の部分のみをテストするため，その箇所以外においてはテストが不十分になる場合がある．

以上の 2.1〜2.4 の方法では，プログラムの誤りを自動的に検出し，それを学習者に指摘することは難しい．

第3章　ソフトウェア開発における自動テスト

一般的なプログラムの自動テストとして用いられている，ソフトウェア開発におけるプログラムの自動テストについて説明する．

ソフトウェア開発における自動テストは，プログラムの動作を詳細に定めた仕様どおりに,プログラムが動作するか確認するテストのことである.これは，仕様をもとに作成されたテストケースに基づいて実施される．テストケースとは，テスト対象とするプログラムの動作エラーを検出するための入力とその入力に対して出力されるべき出力の組のことである．

図１にプログラムの自動テストシステムの構成図を示す．自動テストシステムの入力は，テスト対象の実行ファイルと実行ファイルをテストするために仕様から作成されたテストケース集合である．また，出力は，テスト結果の集合である．テスト結果の集合は，テストケースと実行ファイルの出力，動作チェックの判定結果からなる．

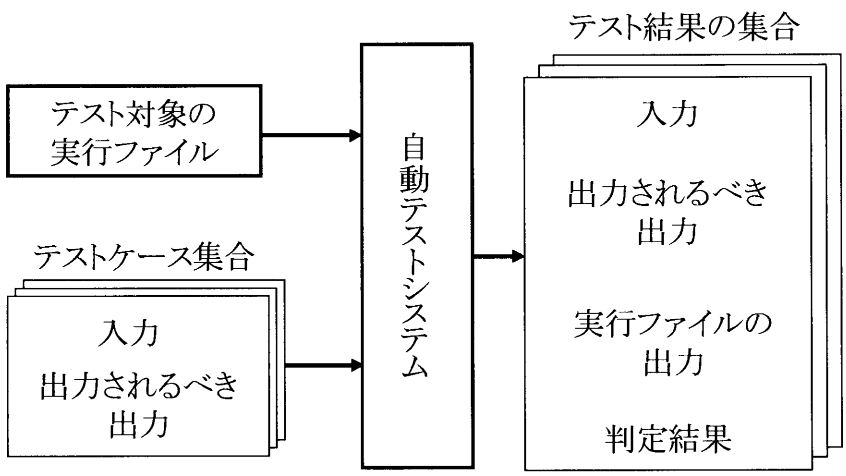

図１　ソフトウェア開発における自動テストシステムの構成

次に，プログラムの自動テストの手順を示す．

(a) テストケースの入力をテスト対象の実行ファイルに入力することで出力を得る．

(b) (a)で得られた出力とテストケースの出力されるべき出力において，それぞれの文字列の一字一句が一致しているかを比較することで，プログラムの動作エラーの有無を判断する．

図 2 にソフトウェア開発におけるプログラムの自動テストの流れを示す．

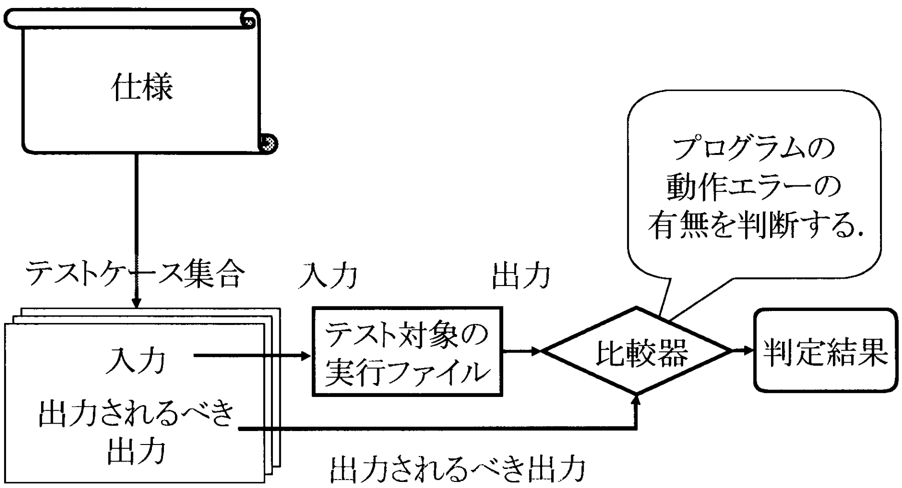

図 2　ソフトウェア開発における自動テストの流れ

第4章　ソフトウェア開発における自動テストの考察

ソフトウェア開発における自動テストを，そのままの形で，プログラミング演習に使用することが可能であるかを考察した．

4.1　プログラミング演習の課題

ソフトウェア開発における自動テストを行うには，図1に示したように，仕様から作成したテストケースを自動テストシステムに入力する必要がある．しかし，プログラミング演習では課題からテストケースを作成することは容易ではない．

プログラミング演習で出題される課題の一例を図3に示す．

三角形の2つの内角の度数から三角形の種別を判別するプログラムを作成せよ．

図3　プログラミング演習で出題される課題の例

図3の一例のように，プログラミング演習で用いられる課題は，入出力の書式が厳密に与えられていないため，課題からテストケースを作成することができない．よって，この自動テストをプログラミング演習で出題される一般的な課題に適用することができない．

4.2　自動テスト可能な課題

自動テストが可能な課題について考察する．自動テストを可能にするために，課題は入出力の書式を厳密に定められたものとなる．図4は入出力の書式が厳密に定められた課題の一例である．

1. 二つの整数値を読み込む.
2. 読み込んだ値のどちらか（または両方）が正でない，あるいは 180 以上の場合には,「1 以上 180 未満の自然数を入力してください.」と表示してプログラムを終了する.
3. 読み込んだ値の和が 180 以上の場合,「内角の和が 180 度以上になりません.」と表示してプログラムを終了する.
4. 三角形の種別（正三角形，二等辺三角形，二等辺三角形，直角三角形）を判別して，以下のメッセージのいずれかを表示してプログラムを終了する.
   - 「正三角形です.」
   - 「直角二等辺三角形です.」
   - 「二等辺三角形です.」
   - 「直角三角形です.」

図 4　入出力の書式が厳密に決められている課題の例

図 4 のような入出力の書式が厳密に決められている課題は，記述量が多いため，初心者はその課題を正確に理解し，プログラムを作成することが難しい．そのため，入出力の書式が厳密に決められている課題は，初心者に対して不向きなものとなる．

以上の考察から，ソフトウェア開発における自動テストをそのままの形でプログラミング演習に適用することはできない．

第5章　プログラムの自動テストを応用したプログラミング演習支援方式

プログラミング演習支援として，字下げが正しく行われているかを指導することを目的とした支援やマジックナンバーを指摘することを目的とした支援などがある．本研究のプログラミング支援方式の目的は，ソフトウェア開発におけるプログラムの自動テストをプログラミング演習に適用できるよう検討し，その検討結果を用いて学習者に動作エラーを気付かせることのできる情報を提供することである．

本提案方式では，あらかじめプログラムの正解例を用意し，それをもとにテスト入力を自動生成する．プログラムの正解例は，課題を満たすように作成されているため，正解例への入力に対して正しい出力の一例が応答として得られる．もちろん，正解例とテスト対象の実行ファイルに同じ入力をすることで得られるそれぞれの出力を用いてテストすることも考えられるが，プログラムの自動テストでは,文字列が一字一句同じでない場合,動作エラーと判断される．そこで提案手法では，プログラムの正解例から生成されたテスト入力を対象とする学習者のプログラムに入力し，その応答を学習者に提示することで，学習者に動作エラーを気付かせる．

今回は，提案手法の可能性を確認すべく，テスト対象とするプログラムの構造を以下のように限定した．

- 制御構造が if 文だけからなるプログラム
- if 文の条件式を構成する等式・不等式が以下の形のプログラム
    - 「変数 比較演算子 定数」
    - 「変数 比較演算子 変数」
    - 「変数 演算子 変数 比較演算子 定数」

提案方式の流れ手順を示す．

(a) プログラムの正解例からテスト入力集合を生成する．
(b) 手順(a)で生成されたテスト入力をテスト対象の実行ファイルに入力し，出力を得る．
(c) テスト入力と手順(b)で得られた出力をリスト化し，動作結果として学習者に提示する．
(d) 学習者は，提示されたリストを見ることでプログラムにおける動作エラーの有無を判断する．

図5に提案方式の流れを示す．

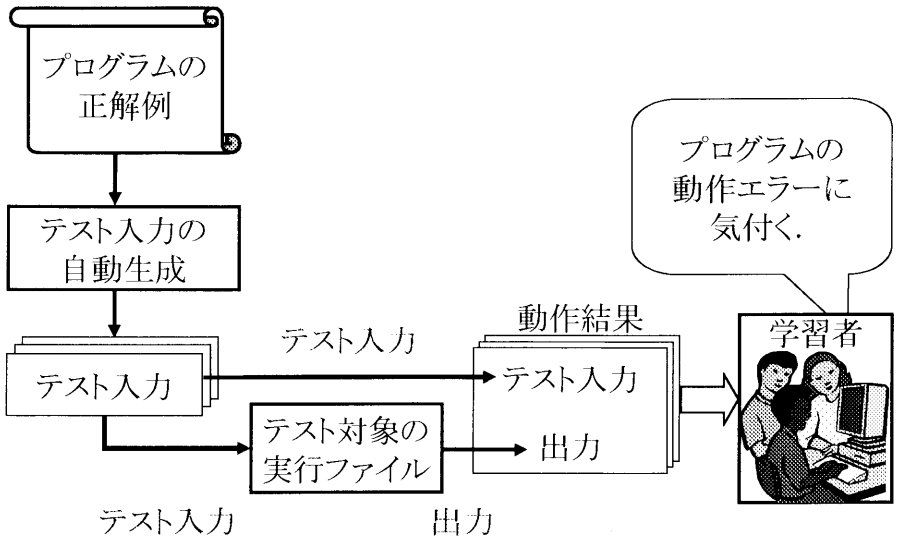

図 5　提案方式の流れ

テスト入力の自動生成を行う際に，if 文の条件式を構成する等式・不等式の形に応じて，テスト入力集合の生成方法の手順を変える．以下にそれぞれの手順と入力生成例を示す．

- **等式・不等式が「変数 比較演算子 定数」の形のとき**

  (a) プログラムの正解例から条件を見つける．
  (b) if 文中の等式・不等式の定数を見つける．
  (c) 定数・定数のすぐ上・定数のすぐ下の値を生成する．
  (d) 変数が(c)で生成した値になるように，プログラムの正解例をフロー解析し，テスト入力を決める．

図 6 は課題の例，図 7 は図 6 の正解例である．この例を用いて「変数 演算子 定数」の例を示す．

テストの点数を読み込み(テストの点数＋10)が 60 以上なら「合格です. 」と表示し, 60 未満なら「不合格です. 」と表示する.

図6　課題例

```
# include < stdio.h >
int main ( void )
{
   int score , num ;
   printf ( “ 点数を入力してください. ¥n “ ) ;
   printf ( “ 点数: “ ) ;
   scanf ( “ %d ”, & score ) ;
   num = score + 10 ;
   if ( num > = 60 ) {
     printf ( “ 合格です. ¥n “ ) ;
   }
   return (0) ;
 }
```

図7　正解例のプログラム

図7におけるプログラムの8行目からifを見つけた．if文中の式から定数が60であることがわかる.定数・定数のすぐ下の値・定数のすぐ上の値を生成し，59，60，61を得た．変数がこの3値となるように，フロー解析すると，式中の変数であるnumはscoreの関数であることがわかった．num = score＋10より，テスト入力が49，50，51と決定した．

表1　テスト入力

テスト入力値
49
50
51

- **等式・不等式が「変数 比較演算子 変数」の形のとき**

(a) プログラムの正解例から条件を見つける．

(b) if 文中の等式・不等式の変数を見つける．

(c) 変数の範囲を調べるために，プログラムの正解例をフロー解析する．

(d) 変数の範囲に収まる値の整数を1つずつランダムに選ぶ．これが最初の変数の候補となる．

(e) ランダムに選んだ値・ランダムに選んだ値のすぐ上・ランダムに選んだ値のすぐ下の値を生成し，その全てを2番目の変数の値とする．

(f) テスト入力候補のうち，変数の範囲外であれば，その変数の値から外す．

(g) (f)で決定した値になるように，プログラムの正解例をフロー解析し，それぞれの変数におけるテスト入力を決める．

図8は課題の例，図9は図8の正解例の一部である．この例を用いて「変数 比較演算子 変数」（図9の4行目の不等式）の例を示す．

サイコロを2回ふり，(1回目の値)＞(2回目の値)となれば「当たり」と表示する．

図8　課題例

```
# include < stdio.h >
int main ( void )
{
     int score1 , score2 ;
     printf ( " 1回目のサイコロの出た目の数を入力してください. ¥n " ) ;
     printf ( " 1回目: " ) ;
     scanf ( " %d " , & score1 ) ;
     printf ( " 2回目のサイコロの出た目の数を入力してください. ¥n " ) ;
     printf ( " 2回目: " ) ;
     scanf ( " %d " , & score2 ) ;
     if ( score1 < 1 || score1 > 6 || score2 < 1 || score2 > 6 ) {
       printf ( " 入力をしなおしてください. ¥n " ) ;
     } else if ( score1 > score2 ) {
       printf ( " 当たり. ¥n " ) ;
     }
     return (0) ;
}
```

図9　正解例のプログラム

図9のプログラムの12行目からelse ifを見つけた．このif文中の式から変数がscore1，score2であることがわかる．変数の範囲を調べるために図9をフロー解析すると，図9の11行目からscore1が1〜6，score2も1〜6の範囲であることがわかった．変数の範囲に収まる値の整数を1つずつ選ぶ．それぞれの値からランダムに選んだのが1と6とする．ここで，この2値が最初の変数の値となる．ランダムに選んだ値・ランダムに選んだ値のすぐ下の値・ランダムに選んだ値のすぐ上の値を生成し，0，1，2，5，6，7を得た．これらの値が2番目の変数の値とする．これらの値が，変数の範囲外であれば，その変数を値から外す．よって，score1の値は1と6，score2の値が1，2，5．6となる．これらの値になるように，プログラムの正解例をフロー解析し，それぞれの変数におけるテスト入力を決めると表2となる．この表のscore1は最初の変数，score2は2番目の変数とする．

表 2　テスト入力

テスト入力値	
score1	score2
1	1
	2
	5
	6
6	1
	2
	5
	6

- **等式・不等式が「変数 演算子 変数 比較演算子 定数」の形のとき**
  - (a) プログラムの正解例から条件を見つける．
  - (b) if 文中の等式・不等式の変数を見つける．
  - (c) 変数の範囲を調べるために，プログラムの正解例をフロー解析する．
  - (d) 変数の範囲に収まる値の整数を 1 つずつランダムに選ぶ．
  - (e) ランダムに選んだ値・ランダムに選んだ値のすぐ上・ランダムに選んだ値のすぐ下の値を生成し，その全てをそれぞれの変数の値とする．
  - (f) 定数・定数のすぐ上・定数のすぐ下の値を生成し，それぞれの変数の値に追加する．
  - (g) さらに，変数の範囲の最も大きい値・小さい値においても，(最も大きい値・小さい値)・(最も大きい値・小さい値のすぐ上の値)・(最も大きい値・小さい値のすぐ下の値) を生成し，それぞれの変数の値に追加する．
  - (h) それぞれの変数の値が，その変数の範囲外であれば，その変数の値から外す．
  - (i) (g)で決定した値になるように，プログラムの正解例をフロー解析し，それぞれの変数におけるテスト入力を決める．

図 10 は課題の例，図 11 は図 10 の正解例の一部である．この例を用いて「変数 演算子 変数 比較演算子 定数」(図 11 の 4 行目の不等式）の例を示す．

サイコロを2回ふり，(1回目の値)＋(2回目の値)＞6となれば「当たり」と表示する．

図10　課題例

```
# include  < stdio.h >
int main ( void )
{
       int score1 , score2 ;
       printf ( " 1回目のサイコロの出た目の数を入力してください. ¥n " ) ;
       printf ( " 1回目: " ) ;
       scanf ( " %d " , & score1 ) ;
       printf ( " 2回目のサイコロの出た目の数を入力してください. ¥n " ) ;
       printf ( " 2回目: " ) ;
       scanf ( " %d " , & score2 ) ;
       if ( score1 < 1 || score1 > 6 || score2 < 1 || score2 > 6 ) {
         printf ( " 入力をしなおしてください. ¥n " ) ;
       } else if ( score1 + score2 > 6 ) {
         printf ( " 当たり. ¥n " ) ;
       }
       return (0)
}
```

図11　正解例のプログラム

図11のプログラムの12行目からelse ifを見つけた．このif文中の式から変数がscore1, score2であることがわかる．変数の範囲を調べるために図11をフロー解析すると，図11の11行目からscore1が1～6，score2も1～6の範囲であることがわかった.変数の範囲に収まる値の整数を1つずつランダムに選ぶ．ランダムに選んだ値が1と6とする．ここで，ランダムに選んだ値・ランダムに選んだ値のすぐ下の値・ランダムに選んだ値のすぐ上の値を生成し，0, 1, 2, 5，6，7を得た．また，式中の定数の値は6なので，式中の定数・定数のすぐ上の値・定数のすぐ下の値を生成すると，5，6，7が生成される．さらに，変数

の範囲の最も大きい値・小さい値においても，(最も大きい値・小さい値)・(最も大きい値・小さい値のすぐ上の値)・(最も大きい値・小さい値のすぐ下の値)を生成すると，0，1，2，5，6，7となる．

これらの値をそれぞれの変数の値に追加すると，0，1，2，5，6，7が得られる．それぞれの変数の値が，その変数の範囲外であれば，その変数の値から外す．よって，それぞれの変数の値がscore1，score2共に，1，2，5，6となる．これらの値になるように，図11をフロー解析し，それぞれの変数におけるテスト入力を決めると，テスト入力が表3となった．この表のscore1は最初の変数，score2は2番目の変数とする．

表3　テスト入力

テスト入力値	
score1	score2
1	1
	2
	5
	6
2	1
	2
	5
	6
5	1
	2
	5
	6
6	1
	2
	5
	6

第6章　実験

本研究で提案する方式の有効性を確認するために，有効なテスト入力が生成できるかを確認するための実験を行った．過去に実施された三重大学工学部電気電子工学科におけるC言語のプログラミングに関する講義「プログラミング演習Ｉ」(2005 年度後期) で使用された課題およびその課題に対して学習者が作成したプログラムを対象とした．

- 演習：　2005 年度後期プログラミング演習Ｉ
- 受講者：電気電子工学科 1 年次 48 名
- 課題：　三角形の種別を判別するプログラム
- 提出されたプログラムの数：　18 個
- 誤りのあるプログラムの数：　16 個／18 個中

この演習で出題された課題を図 12 に示す．

1. 二つの整数値を読み込む．変数名は angle1，angle2 とすること．
2. 読み込んだ値のどちらか（または両方）が正でない，あるいは180 以上の場合には，そのことを表示してプログラムを終了する．
3. 読み込んだ値の和が 180 以上の場合，そのことを表示してプログラムを終了する．
4. 三角形の種別（正三角形，直角二等辺三角形，直角三角形，二等辺三角形）を判別し，そのことを表示してプログラムを終了する．

図 12　三角形の種別を判別するプログラム作成の課題

図 12 の課題から作成した正解例を図 13 に示す．図中の①～⑥は if 文の条件式を構成する等式・不等式が存在する箇所である．

```
/* 三角形の種別を判別するプログラム */

int main ( void ) {
    int angle1, angle2 ;
    printf ( " 三角形の１つ目の内角を入力して下さい. ¥n " ) ;
    printf ( " １つ目の内角：" ) ;
    scanf ( " %d ", & angle1) ;
    printf ( " 三角形の２つ目の内角を入力して下さい. ¥n " ) ;
    printf ( " ２つ目の内角： " ) ;
    scanf ( " %d ", & angle2) ;

    if ( angle1 < = 0 || angle1 > = 180 ||
    angle2 < = 0 || angle2 > = 180) {
        /* ① 三角形の内角を表す数値として不適切な場合 */
        printf( " 1以上180未満の自然数を入力してください. ¥n " ) ;
    } else if ( angle1 + angle2 > = 180 ) {
        /* ② 入力された二角の和が180度以上の場合 */
        printf( " 三角形になりません. ¥n " ) ;
    } else { ;

        if ( angle1 == 60 && angle2 == 60 ) {
            /* ③ 正三角形になる場合 */
            printf ( " 正三角形です. ¥n " ) ;
        } else if (( angle1 == 45 && angle2 == 45) ||
        ( angle2 == 45 && 180 - angle1 - angle2 == 45 ) ||
        ( 180 - angle1 - angle2 == 45 && angle1 == 45 )) {
            /* ④ 直角二等辺三角形になる場合 */
            printf ( " 直角二等辺三角形です. ¥n " ) ;
        } else if ( angle1 == angle2 || angle2 == 180 - angle1 - angle2 ||
        180 - angle1 - angle2 == angle1 ) {
            /* ⑤ 二等辺三角形になる場合 */
            printf ( " 二等辺三角形です. ¥n " ) ;
```

図13-1　プログラムの正解例（1/2）

```
        } else if ( angle1 == 90 || angle2 == 90 ||
        180 - angle1 - angle2 == 90 ) {
            /* ⑥ 直角三角形になる場合 */
            printf ( " 直角三角形です. ¥n " ) ;

        } else {
            /* 不等辺三角形になる場合 */
            printf ( " 不等辺三角形です. ¥n " ) ;
        }
    }
    return (0);
}
```

図 13-2　プログラムの正解例（2/2）

表 4 と 5 に,プログラムの正解例である図 13 から自動生成したテスト入力を示す．それぞれの表の上に表示した①～⑥は，図 13 の if 文の条件式を構成する等式・不等式が存在する箇所（①～⑥）から生成したテスト入力である．例として図 13 の条件式①と②のテスト入力を生成する手順を示した．

① angle1 < = 0 || angle1 > = 180 || angle2 < = 0 || angle2 > = 180

図 13 におけるプログラムの 7 行目から if を見つけた. 条件式中から定数が 0 と 180 であることがわかる. 定数・定数のすぐ下の値・定数のすぐ上の値を生成し, -1, 0, 1, 179, 180, 181 を得た. これらの値が変数（angle1, angle2）の値となるようにフロー解析すると, 入力する変数も angle1 と angle2 であることがわかった. よって, -1, 0, 1, 179, 180, 181 を組み合わせたテスト入力となる. 実際に生成したテスト入力は表 4 の左になる.

② angle1 + angle2 > = 180

図 13 におけるプログラムの 10 行目から else if を見つけた．この条件式中から変数が angle1 と angle2 であることがわかる．変数の範囲を調べるために図 13 をフロー解析すると，図 13 の 7 行目から angle1 が 1～179，angle2 も 1～179 の範囲であることがわかった．変数の範囲に収まる値の整数を 1 つずつランダムに選ぶ．ランダムに選んだ値が 40 と 78 とする．ここで，ランダムに選んだ

値・ランダムに選んだ値のすぐ下の値・ランダムに選んだ値のすぐ上の値を生成し，39，40，41，77，78，79 を得た．また，式中の定数の値は 180 なので，式中の定数・定数のすぐ上の値・定数のすぐ下の値を生成すると，179，180，181 が生成されることとなる．さらに，変数の範囲の最も大きい値・小さい値においても，(最も大きい値・小さい値)・(最も大きい値・小さい値のすぐ上の値)・(最も大きい値・小さい値のすぐ下の値) を生成すると，0，1，2，178，179，180 となる．これらの値をそれぞれの変数の値に追加すると，0，1，2，39，40，41，77，78，79，178，179，180 が得られる．それぞれの変数の値が，その変数の範囲外であれば，その変数の値から外す．よって，それぞれの変数の値が angle1，angle2 共に，1，2，39，40，41，77，78，79，179 となる．これらの値になるように，図 11 をフロー解析し，それぞれの変数におけるテスト入力を決めると，テスト入力が表 4 の中央になる．この表の angle1 は最初の変数，angle2 は 2 番目の変数とした．

表４　正解例から生成されるテスト入力（①～②）

①

テスト入力値	
angle1	angle2
-1	-1
	0
	1
	179
	180
	181
0	-1
	0
	1
	179
	180
	181
1	-1
	0
	1
	179
	180
	181
179	-1
	0
	1
	179
	180
	181
180	-1
	0
	1
	179
	180
	181
181	-1
	0
	1
	179
	180
	181

②

テスト入力値		テスト入力値		テスト入力値	
angle1	angle2	angle1	angle2	angle1	angle2
1	1	41	1	178	1
	2		2		2
	39		39		39
	40		40		40
	41		41		41
	77		77		77
	78		78		78
	79		79		79
	178		178		178
	179		179		179
2	1	77	1	179	1
	2		2		2
	39		39		39
	40		40		40
	41		41		41
	77		77		77
	78		78		78
	79		79		79
	178		178		178
	179		179		179
39	1	78	1		
	2		2		
	39		39		
	40		40		
	41		41		
	77		77		
	78		78		
	79		79		
	178		178		
	179		179		
40	1	79	1		
	2		2		
	39		39		
	40		40		
	41		41		
	77		77		
	78		78		
	79		79		
	178		178		
	179		179		

表 5　正解例から生成されるテスト入力（③～⑥）

③

テスト入力値	
angle1	**angle2**
59	59
	60
	61
60	59
	60
	61
61	59
	60
	61

④

テスト入力値	
angle1	**angle2**
44	44
	45
	46
	89
	90
	91
45	44
	45
	46
	89
	90
	91
46	44
	45
	46
	89
	90
	91
89	44
	45
	46
90	44
	45
	46
91	44
	45
	46

⑤

テスト入力値	
angle1	**angle2**
27	26
	27
	28
	71
	72
	73
72	26
	27
	28
	71
	72
	73

⑥

テスト入力値	
angle1	**angle2**
89	89
	90
	91
90	89
	90
	91
91	89
	90
	91

表 6 は，動作エラーの原因が含まれる学習者のプログラムを，動作エラーの原因別に著者が分類したものである．動作エラーに対して，そのエラーを含むプログラムを提出した人数とそのエラーが検出可能かどうかを示した．

表6　提出されたプログラムの分類

動作エラーの原因	人数[人]／18人中	動作エラー検出可能
① 読み込んだ値が正でない あるいは180以上でない場合 例 ： angle1 <= 0 && angle1 >= 180 （ ‖が &&になっている．）	5	○
② 読み込んだ値の和が180以上の場合 例 ： プログラムに含まれていない．	9	○
③ 正三角形の場合	0	－
④ 直角二等辺三角形の場合 例 ： ⑤や⑥よりも先に書くべきだが それらの後に書かれている．	10	○
⑤ 二等辺三角形の場合 例 ： angle1 == angle2 （2つの角度の条件しか定義していない）	6	○
⑥ 直角三角形の場合 例 ： angle1 == 90 ‖ angle2==90 （2つの角度の条件しか定義していない）	4	○

表6より，それぞれの動作エラーを学習者に気付かせることのできる情報を作成できたことがわかる．

第7章　まとめ

現状では，プログラムの作成に慣れていないため，十分なテストをせずに提出する学習者も多い．また，受講者が多人数となると，講師が演習時間中に学習者の作成した個々のプログラムに対してきめ細やかなコメントを返すことは難しい．修正すべき点を具体的に指摘されないため，プログラムに動作エラーがあっても，それに気付かない学習者がいる．プログラミングの初心者の場合には，特にこれが顕著になる．

以上のことから，２章で示すようなシステムを用いた研究が行われている．しかし，このような方法であれば，システムが学習者のプログラムに存在する動作エラーを指摘できなかったり，システムに適用し易いように課題を対応させるため学習者が課題を理解することができなかったりと,問題点が存在する．

本研究では，課題で指定されたプログラムの正解例を用意し，それをもとにプログラムの提出直後に動作エラーを学習者に提示する方法を提案した.また，提案する方法を用いて人手でテスト入力集合の生成を行った．生成した個々のテスト入力を用いて学習者が作成したプログラムを実行したところ，動作エラーを起こすプログラム全てに対して，動作エラーがあることに気付くことのできる出力が得られた．

この結果，本研究で提案する方式によって個々の学習者に自分のプログラムの動作エラーに気付くことができる情報をプログラムが提出された直後に提供することができることが確認できた．

参考文献

[1] 中澤達夫，和崎克己，師玉康成，「プログラミングレポート自動評価における事前処理」，情報処理学会・電子情報通信学会共催，情報科学技術フォーラム，（FIT2002）論文集 pp. 247-248，2002

[2] 小西達裕，鈴木浩之，伊東幸宏，「プログラミング教育における教師支援のためのプログラム評価機構」，電子情報通信学会，電子情報通信学会論文誌，Vol.J83-D-I No.6，pp682-692，2000

[3] 櫻井桂一，「プログラミング実習を支援するためのＣプログラム誤り検出システムの開発」，日本教育工学会，日本教育工学会誌，Vol.25，pp. 67-70，2001

[4] 望月将行，森田直樹，高瀬治彦，北英彦，林照峯，「自動テスト機能を備えたプログラミング演習支援システム」，コンピュータ利用教育協議会，PCカンファレンス，pp.52-55，2004

[5] Boris Beizer，「ソフトウェアテスト技法」，日経BP出版センター，1994

業績一覧

[i] 小坂祐章，北英彦，高瀬治彦，林照峯，「プログラミング初心者のためのテスト入力値自動生成法」，計測自動制御学会，計測自動制御学会，平成18年度三重地区計測制御研究講演会講演論文集，P-19，2006

[ii] 小坂祐章，平井淳之，「構造可変型ロボットシステムにおける故障対応の研究」，電気関係学会，平成17年度電気関係学会東海支部連合大会講演論文集，O-536，2005

[iii] 小坂祐章，平井淳之，「構造可変型ロボットにおける電力伝達に関する研究」，計測自動制御学会，平成17年度三重地区計測制御研究講演会講演論文集，A-6，2005

謝辞

本論文は，著者が三重大学大学院工学研究科博士前期課程に在籍中に行った研究をまとめたものである．本研究を進めるにあたり，懇切丁寧な御指導と御督励を賜った三重大学工学部電気電子工学科林照峯教授，北英彦助教授，高瀬治彦助手に感謝の意を表します．

また，本研究論文をまとめるにあたり，多くのご指導を頂きました計算機工学研究室の皆様方,査読をしていただいた森香津夫助教授に深く感謝致します．

最後に，何不自由なく大学生活を送らせて頂いた両親に心から深く感謝致します．