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

望月将行　森田直樹　高瀬治彦　北英彦　林照峯
三重大学大学院 工学研究科
motizuki@hayashi.elec.mie-u.c.jp

1. はじめに

プログラミング演習では,講師が学習者の作成したプログラムに対して問題点の指摘を行い,学習者は講師から指摘された問題点を修正しプログラムを再提出する.この講師と,学習者の修正と再提出を繰り返すことが理想的なプログラミング演習である.しかし,現状では,講師が学習者の作成した個々のプログラムに対してきめ細かなコメントを速やかに返すことは手間がかかるために難しい.そこで,多人数講義においても個々の学習者にきめ細かなコメントを返すことができるように,プログラミング演習における講師の負担を軽減することを目的としたシステムを提案する.

２．プログラミング演習の流れと問題点

プログラミング演習では,学習者にプログラミング能力を身につけさせるためにプログラムの作成を行わせる.

理想的なプログラミング演習の流れは以下のようになる.

(1) 講師が学習者にプログラム作成の課題を提示する.
(2) 学習者は課題で指示されたプログラムの作成・テスト・デバグを行い,完成したプログラムを講師に提出する.
(3) 講師は,学習者の作成したプログラムのソースコードの確認および実行プログラムの動作確認を行い,個々の学習者に対してその学習者のプログラムの問題点を指摘する.
(4) 学習者は指摘された問題点に対してプログラムの修正を行う.
(5) (2)～(4)の作業を問題点がなくなるまで繰り返す.

しかし,学習者が多数になると,講師の負担が膨大なものとなる.現状では,プログラムを受け取った後,典型的な間違いや解答例を講義中に説明するというやり方をとることが多い.この方法では,提出したプログラムに間違いや修正すべき点があることに気づくことができない学習者がいる.特にプログラミングの初心者ではそれが顕著になる.また,コンパイルができない・動作がおかしいプログラムであってもそれを指摘されないので,十分にテストやデバグをせずにプログラムを提出する学習者がいる.このような状況では,学習者はプログラミングを効果的に学習することができない.さらに,コンパイルができない・明らかに動作がおかしいプログラムが提出されるような状況では,本来は学習者が行うべきことを講師が行わなければならなくなり,ますます講師の負担が増加する.

３．講師の負担を軽減する方法

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.それらの研究の中で,学習者が作成したプログラムの誤りを自動的に検出するために以下のような手法がとられている.

(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[1][2].テストケースとは,プログラムにテストのために入力する値とプログラムが正しく動作したときに出力されるべき値の組である.
(2) 課題に対する標準的なプログラム,あるい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[3][4].
(3) プログラミングの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[5].

本研究では,(1)の方法によりプログラムの誤りを検出する.この方法は,ソフトウェア開発で行われているプログラムのテストに用いられており,プログラムの品質が保証されると考えられる.また,プログラミング演習支援システムには,(1)の方法にて従来から提案し用いられているプログラムの受理とコンパイルチェックの自動化も取り入れる.この提案するシステムを用いることで,システムが自動でコンパイルチェックとテストケースを用いた動作確認を行うので,学習者から提出されるプログラムの品質が保証される.

講師は,システムによる自動テストのエラーを解決できない学習者に対して,的確なアドバイスを行えばよい.また,エラーがない,エラーを解決した学習者には,システムでは見つけることができなかった,より複雑な間違いやプログラムの読みやすさ,分かりやすさを向上させるための修正すべき点の指摘のみを行えばよい.これによって,講師は個々の学習者に対してよりきめ細かなコメントを返すことができるようになる.

しかし,本研究が対象としている学習者はプログラミングの初心者であり,プログラミング演習にお

いて課される課題は,出力の値を厳密に指定することはほとんどなく，指定されても，その出力値の形式を満たすプログラムを作成することが初心者には難しい[1][2]．例を以下に示す．

【例題 1】

フィボナッチ関数 fib(n)を作成し，入力された整数を n としたフィボナッチ数列を求めるプログラムを作成せよ．n は正の整数が与えられ，関数の定義は，n=0 のとき 0，n=1 のとき 1，n がそれ以外の場は，fib(n-1)+fib(n-2)である．配列は用いず，関数の再起を使用すること．

プログラムは正の整数を入力してもらうために「正の整数を入力してください：」と表示し，「：」の後に正の整数を入力してもらう．結果は，例えば入力値が 5 の場合は，「fib(5)=8」とする．

例題 1 の要点 : 関数の再起が使用できるかを確認する．

<テストケース>
[入力値]　5
[出力値] 正の整数を入力して下さい：fib(5)=8

このテストケースを用い, プログラムから出力される値とテストケースの出力値を比較し, 動作確認を行う方法では, 出力される値がテストケースの出力値と完全に一致しなくてはならない．そこで，プログラミングの初心者向けのために, 出力を厳密に指定することなく, 出力される値を用いて動作確認を行う方法を提案する．この動作確認では，あらかじめ出力される値に含まれるべき文字列を指定し，出力された値の中でその文字列の有無を判断する方法をとる．なお，含まれるべきでない文字列の指定も行えるようにする. 指定する文字列の有無を判断する際には，空白と改行の有無，大文字小文字と半角全角による違いを無視して判断するものとする．この方法では，動作確認に用いるテストケースは，3.(2)で説明した従来のものと違い，入力値と出力される値に含まれるべき文字列, 含まれるべきでない文字列の組となる．（以下，テストケースはこの定義にて用いる．）

提案する方法を，例題 1 で説明すると，入力値が「5」の場合，含まれるべき文字列は「8」となる．偶然にも出力される値に「8」が含まれる場合も考えられるので, テストケースを複数用意する, また, 出力される値に条件を付けて含まれるべき文字列を「=8」や「fib(5)=8」等を指定するといった方法で，動作を確認することができる．

例題 1 を提案する方法で用いた場合に修正すると以下になる．例題 1 に比べ，出力される値に制約が緩く，また，空白の有無と全角半角の違いが含まれていても,「fib(5)=8」と判断するので，初心者でもこの制約を満たすプログラムを作成できると考えられる．

【例題 1 修正版】

フィボナッチ関数 fib(n)を作成し，入力された整数を n としたフィボナッチ数列を求めるプログラムを作成せよ．n は正の整数が与えられ，関数の定義は，n=0 のとき 0，n=1 のとき 1，n がそれ以外の場は，fib(n-1)+fib(n-2)である．配列は用いず，関数の再起を使用すること．

ただし，出力されるフィボナッチ数列は「fib(5)=8」と表示すること．(入力が 5 の場合)

次節にて動作確認の方法を例題 2 を用いて説明する．

４．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

講師にかかる負担を軽減するために, コンパイルチェック, 動作確認をシステムが自動的に行う．システム構成を図３に示す．

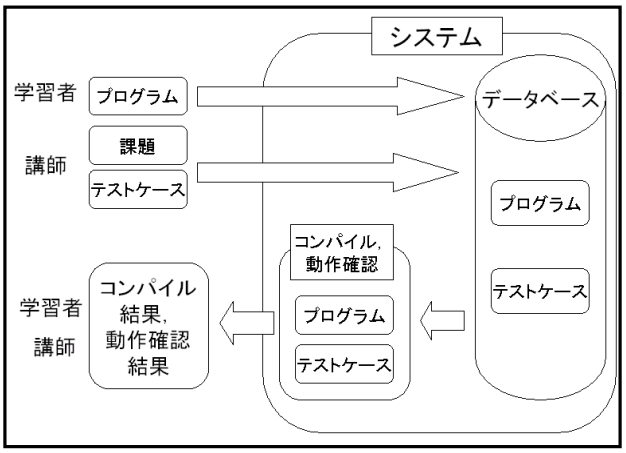

図 3　システム構成

システムは, 学習者から提出されたＣ言語プログラムと講師から入力された動作確認に用いるテスト値を受け取り, データベースに保存する. その後, 保存されたＣ言語プログラムをコンパイルして, コンパイルエラーがあった場合はそのエラーを学習者と講師に伝える. コンパイルにエラーがなかった場合は, Ｃ言語プログラムとテスト値を用いてのプログラムの動作確認を行い, 動作確認の結果を学習者と講師に示す.

以下に示す例題 2 を用い, 実装した画面を用いてその流れを説明していく．

【例題 2】

100 個の要素からなる一次元配列 a を宣言し，すべての a[i] に対して，i の値を設定し，配列 a の値を a[11]=11 のように全て表示するプログラムを作成せよ. ただし, for 文を使用し，要素数は#define 指令で定義して使用すること．

例題 2 の要点：配列の要素は 0 から始まり，100 個の要素とは，a[0]～a[99]を意味する．この問題の目的は，学習者がそれを理解し，配列の指定を a[1]～a[100]ではなく，a[0]～[99]と指定できているかを確認することとする．

4．1　プログラムの受理

動作確認とコンパイルチェックをシステムが自動で行うためには，提出されたプログラムを計算機に読み込ませる必要がある．講師がこれを行うのは負担であるため，プログラムの受理には Web を用いる．これにより，学習者から提出されたプログラムを計算機上のデータとして，かつ自動で受理できる．システムは提出されたプログラムを管理するために，作成者や提出日などの情報と共にデータベースにこれを保存する．学習者がプログラムを提出する際の提出画面を図 4 に，プログラムの提出完了画面を図 5 に示す．

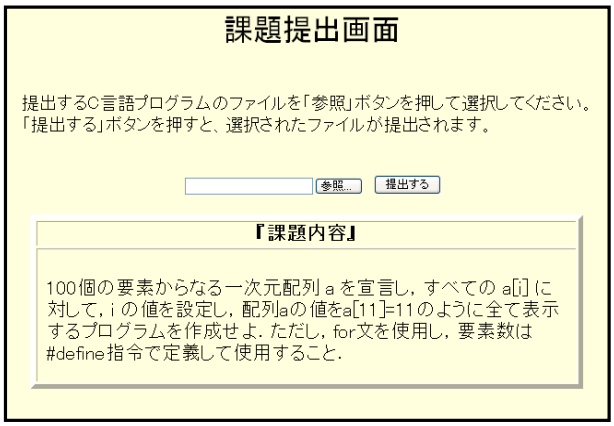

図 4　プログラム提出画面（学習者）

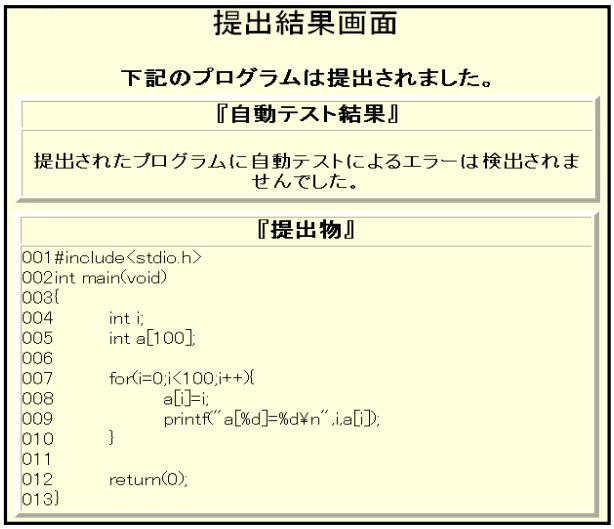

図 5　プログラム提出完了画面（学習者）

図 4 の画面中央にある「参照」ボタンを押すことにより，提出するプログラムファイルを選択でき，右隣にある「提出」ボタンを押すことで選択中のファイルを提出することができる．また，現在提出中の課題名が画面上部に課題内容が画面中部に表示されている．過去にプログラムを提出したことがある場合は，過去に提出したプログラムが画面下部に表示される．

図 5 では，画面上部にプログラムの提出が完了したメッセージが，画面中部に学習者が提出したプログラムが表示され，画面下部に自動テスト結果が表示される．自動テストの結果は，コンパイルチェックの結果と動作確認の結果からなる．これらについては，次節以降で詳しく述べる．

4．2　コンパイルチェック

次にシステムは提出直後にプログラムのコンパイルチェックを行う．データベースからプログラムを読み込み，コンパイラにかける．コンパイルエラーがあった場合には，直ちにそのエラーを学習者に伝える．ただし，コンパイルが出力するエラーメッセージは初心者にはわかりにくいので，初心者でもわかるような表現を添えて表示する．また，エラーメッセージだけでなく，そのエラーに対してデバグを行うときに参考となる初心者向けのヒントを付与する．また，講師がプログラムを閲覧するときにもコンパイル結果を付与する．コンパイルチェック後にコンパイルエラーがあった場合の表示画面を図 6 に示す．

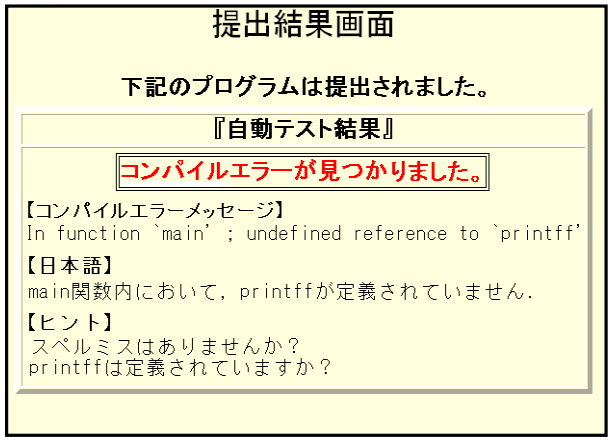

図 6　コンパイルチェック語の表示画面（学習者）

画面上部には課題に関する情報が，画面中部にはコンパイルチェックの結果が表示される．また，画面下部には提出されたプログラムが表示される．コンパイルエラーメッセージが「コンパイラからのメッセージ」の欄に表示され，初心者でもわかるような表現がその下に添えられている．また，初心者向けのヒントがコンパイラからのメッセージの下に表示されている．図の例では，提出されたプログラム中 printf 関数がスペルミスで「printff」と記入されているので，「In function `main'; undefined reference to `printff'」とコンパイルエラーが出力されている．また，初心者でもわかるような表現として，「main 関数内において，printff が定義されていません．」を，ヒントとして，「スペルミスはありませんか？」「printff は定義されていますか？」というメッセージが付与されている．

学習者は, このメッセージがプログラム提出時に提示されることにより, 講師に負担をかけずにその問題点に気づくことができ, 修正する機会を得ることができる.

4. 3 動作確認

プログラムのコンパイル時にエラーが出力されなければ, コンパイル時に作成されたプログラムの実行プログラムを用いて動作確認を行う.

提出されたプログラムの動作確認において, 例題 2 の要点を満たすかを判断する場合, 出力されるべき値は, 図１０の左側になる. この出力値の中で, 「a[0]=0」が含まれるかを確認すれば, そのプログラムが例題 2 の要点を満たすかを判断できるので, 講師は出力値に含まれるべき文字列には「a[0]=0」を指定する. また, 含まれるべきでない文字列には「a[100]=100」を指定する. なお, 指定された文字列が「a[0]=0」の場合において, 「a[0] = 0」のような, その文字列に空白が含まれる文字列でも, 「a[0]=0」が含まれると判断するようにする.

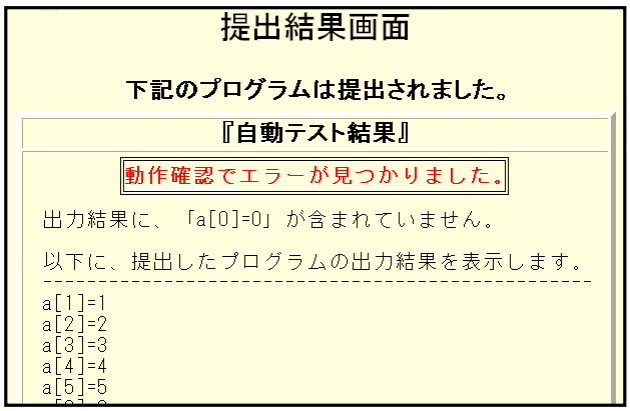

図 9 では, 画面上部にプログラムの提出が完了したメッセージが, 画面中部に動作確認の結果が表示される. また, 画面下部には学習者が提出したプログラムが表示される.

5.プログラムの動作確認結果

4 節で示した例題 2 を 2004 年度の三重大学工学部電気電子工学科 2 年生 98 名が受講する C 言語プログラミング演習に対して課題として課し, 提出されたプログラムに対して 4.3 節で提案した方法で動作確認を行った. 4.3 節で示した通り, 出力される値に含まれるべき値には「a[0]=0」を指定し, 含まれるべきでない値には「a[100]=100」を指定した. この結果を表 1 に示す.

A[0]=0 が含まれていない	28 名
A[100]=100 が含まれている	27 名
提出した合計人数	98 名

表 1 プログラムの動作確認結果

表 1 より 98 名の学習者のうち, 動作確認で出力された値に「a[0]=0」が含まれていなかった学習者が 28 名いたことがわかる. また, 例題 2 の要点を満たしていなかった出力例は図１０の右側に示す.

出力されるべき値

```
a[0]=0
a[1]=1
a[2]=2
 ・
 ・
 ・
a[98]=98
a[99]=99
```

多かった間違い

```
a[1]=1
a[2]=2
a[3]=3
 ・
 ・
 ・
a[99]=99
a[100]=100
```

図 10 出力されるべき値と多かった間違い

この動作確認の結果, 「a[0]=0」が含まれなかった学生 28 名に対し, 提出時に「a[0]=0」が出力された値に含まれなかったことを提示すれば, 提出したプログラムに間違いがあったことに気づくことができる. また, 講師が提出されたプログラムを閲覧する際にも提示することにより, 講師の負担を軽減できると考えられる.

6. まとめ

プログラミング演習におけるプログラムの動作確認で, 出力される値に出力値に含まれるべき文字列が含まれているかを確認する方法を提案した. 従来の出力される値を厳密に指定する方法と違い, 出力に関する制約が緩やかなので初心者でもこの制約を満たすプログラムを作ることができるのではないかと考えられる.

今後の課題は, 提案した動作確認を取り入れたプログラミング演習支援システムを実際のプログラミング演習にて運用し, 有効性を確認することである.

参考文献

[1] 望月将行, 森田直樹, 高瀬治彦, 北英彦, 林照峯, 「プログラミング演習における自動テスト機能を備えた課題提出システム」, 三重地区計測制御研究講演会, pp.121-124(2003)

[2] 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」, 情報科学技術フォーラム, pp. 247-248 (2002)

[3] 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」, 電子情報通信学会論文誌, Vol. J53-D-I, pp. 682-691 (2000)

[4] 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」, 日本教育工学会誌, vol. 25, pp. 67-70 (2001)

[5] 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」, 日本教育工学会誌, vol.25, pp.67-70 (2001)