Vol.2010-CE-103 No.10
2010/3/6

競争型学習を取り入れた入門的Cプログラミング演習
－ 実行テスト系列による部分採点のための柔軟な照合機能 －

川崎 慎一郎, 富永 浩之

香川大学工学部 〒761-0396 香川県高松市林町 2217-20
E-mail: s05t287@stmail.eng.kagawa-u.ac.jp

あらまし これまで，大学情報系学科の入門的 C 授業において，初心者向けの小コンテスト形式でのプログラミング演習を提案し，大会運営を支援するサーバ tProgrEss を開発してきた．システムでは, 提出されたソースコードの正誤を自動判定し, コンテストの進捗状況を即時に通知する．中間目標として，部分的な仕様に基づく実行テスト系列を用意し，段階的な実装を誘導する．採点機能の改良として，複数の入出力サンプルによる実行結果との柔軟な照合を実現する．

キーワード 初級 C 言語演習，プログラミングコンテスト，実行テスト系列, 出力データ照合

Support for Introductory C Programming Exercise with Contest Style as Competitive Learning
- Matching Function for Partial Scoring in Execution Test Series -

Shinichiro KAWASAKI, Hiroyuki TOMINAGA

Faculty of Engineering, Kagawa University 2217-20 Hayashi, Takamatsu, Kagawa, 761-0396 Japan
E-mail: tominaga@eng.kagawa-u.ac.jp

Abstract To improve introductory C language lesson in computer engineering college, we have proposed programming exercises with a small contest style adjusted to beginners in classroom. It raise students' motivation and activity by competitive learning approach. We have also developed tProgrEss, the contest management Web server. The server judges an uploaded student's program by an execution test with input and output data. We offer several preparation tests, which are for step-by-step sub-goals as partial specification. They give clues and guidelines for solving the final goal. However, the method takes costs in preparing many data and different codes. We revised the judging module with flexible matching for diverting data and code to other tests.

Keyword Introductory C language exercise, Programming contest, Execution test series, Output data match

1. はじめに

大学の情報系学科の多くは，初年次にC言語の入門的な科目を必修とし, 演習を重視している．授業の前半で文法や例題を解説し，後半に類題を解く時間を設ける形態が多いが, 学生の理解度や受講態度に差があり, 全ての学生を集中させて演習に取り組ませることは難しい. そこで，プログラミング演習にも，競争型学習(Competitive Learning)を導入することが注目されている[1]．ゲーム感覚のイベントとして，コンテスト形式を採用すれば, 勝つという目的意識や，学生同士の対抗意識が，学習意欲を高めることが期待される．しかし，一般的なコンテストの方法では，初心者への敷居が高く，入門科目で授業時間内に組み込みにくい.

本研究では，初級C授業と初心者に適応した,

小コンテスト形式のプログラミング演習を提案している[2][3][4][5]．大会運営サーバtProgrEssを開発し，実際の授業で運用している．コンテストの実施形態として，教室型と宿題型の両者を取り入れる．教室型は，毎回の授業の最後に，総まとめ的な位置付けとして，教室内で一斉に実施する．基本的な事項で，例題の類題として，15〜30分程度で解ける問題を主に出題する．宿題型は，1〜2週間の期間内に，学外からのアクセスも許容して行う．応用的な事項の組合せで，60分程度の問題を主に出題する．

小コンテスト形式の演習の進行は，図1の通りである．授業内容に関連する複数の問題を与え，何問かを選択し解答する．問題文に沿ったプログラムをローカルPCで作成し，大会運営サーバにアップロードして正誤の自動判定を行う(図2)．制限時間内に早く多く解いた方が高得点となる．進捗状況はWeb上に公開され，学生が相互に確認できる．学生は，これらの情報を基に，解答の修正や次のテストに取り組む．また，他人の進捗状況を見て，競争意識を高め，意欲を持続させる．

このような演習により，学生は自分のペースで解答を進められる．さらに，学生の提出状況を閲覧する教師監視ページも用意する．自動判定では見逃される不十分なコードや理解不足の点に注意を与える．これにより，解答が進まない学生を早期に発見し，個別に指導する．

2. プログラムの正誤判定

2.1. 解答としてのプログラムの評価方法

学生の作成したプログラムを評価する際，幾つかの段階が考えられる．1つ目は，問題の仕様を満たし，想定される入力に対して，常に正しい出力結果が得られることである．2つ目は，適切な算法に基づいて，効率的な処理が実現されていることである．3つ目は，人間が読むソースコードとして，処理の流れが明確で，誰が見ても理解しやすいことである．1つ目の評価では，ブラックボックステスト，ホワイトボックステストを用い，ほぼ客観的に判定できる．2つ目の評価は，大きなデータを与えて計算時間を測定したり，オーバーフローとなる限界を調べることで，ある程度は推測できる．3つ目の評価は，基準が曖昧で主観的な部分があり，出題意図を理解した教師が自分で採点する必要がある．

2.2. 実行結果と入出力サンプルによる正誤判定

本研究では，演習における判定結果の即時通知を重視するため，1 つ目の評価を採用する．すなわち，事前に用意された入出力サンプルによる実行テストを用いて，プログラムの正誤判定を行う．これをサーバ側に提出されたソースコードに対して行う．まず，アップロードされたソースコードを，システムがコンパイルする．生成された実行バイナリに対し，入力サンプルを与えて実行する．その実行結果と，正解となる出力サンプルとで照合を行う．出力サンプルは，問題文の仕様を満たした模範プログラムによる実行結果である．

照合に成功し，正答と判定されれば，規定のルールに従って計算された得点を与える．ただし，実行結果のみに基づいて照合を行うため，ソースコードの内部については問わない．必要があれば教師が後から自分で内容を吟味する．また，不正コピーなどを防ぐため，ソースコードの類似度を判定するモジュールを組み込む．

2.3. 判定結果の 6 段階

ソースコード提出の判定結果は，図 3 の 6 段階で行う．一般的なオンラインジャッジシステムでは，これら以外に，メモリ使用量など，より具体的な判定段階が設けているものもある[6]．しかし，本研究では，対象が入門的な演習のため，最低限のものに留めている．

(1) 不正提出　提出したコードのファイルサイズや拡張子に不備があることを示す. 誤って実行バイナリを提出した場合などである. また, プロセス制御文のような, 字面で分かる危険コードを排除する.

(2) 静的エラー　コンパイルに失敗し，実行バイナリが生成されなかったことを示す. タイプミスや文法的な間違いである.

(3) 実行時エラー　プログラムの実行がエラーとなり，異常終了したことを示す．主に，配列の範囲外参照やポインタ関連のミスである.

(4) 実行打切　指定時間内にプログラムの実行が終了しなかった，または，出力結果が指定サイズを超えたことを示す. 無限ループが発生して暴走している, 仕様にないデータを要求して入力待ちの状態になっている, と推測される.

(5) 誤答　実行は正常に終了したが，実行結果が出力サンプルとの照合に失敗したことを示す．アルゴリズムや出力書式のミスである.

(6) 正答　正しく実行が終了し，実行結果が出力サンプルとの照合に成功したことを示す.

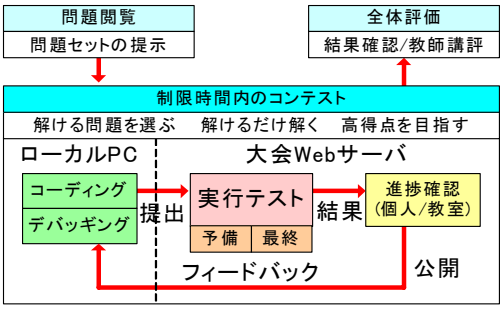

図 1 小コンテスト形式の演習の進行

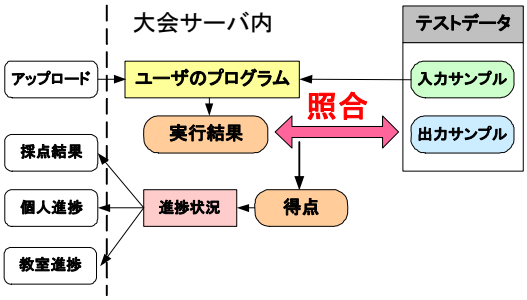

図 2 プログラムの正誤判定の手順

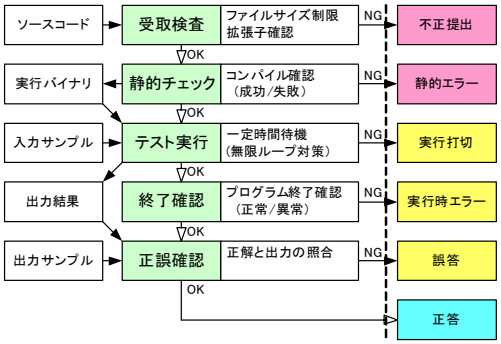

図 3 正誤判定の段階と手順

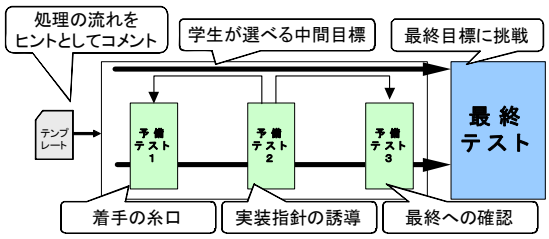

図 4 実行テスト系列の予備テストと最終テスト

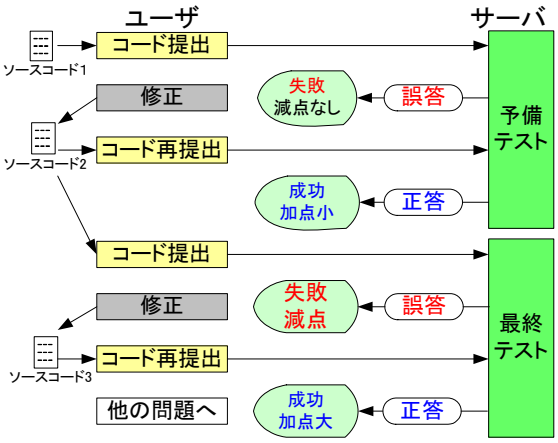

図 5 予備テストと最終テストによる判定の相違

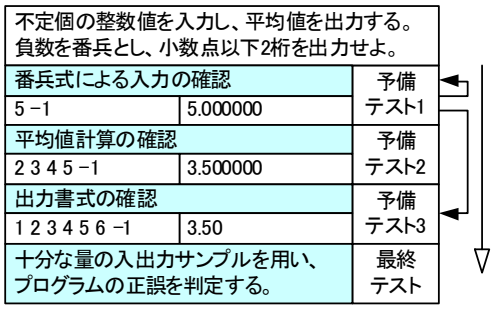

図 6 実行テスト系列の入出力サンプル

予備テスト1：入力値をデータとして正しく格納	
入力の順序とデータ型 不定個入力の番兵での打切り	不適な入力範囲の確認 入力値の計数と振分け

予備テスト2：主要な反復処理や配列操作	
反復の構造(パス、ネスト) 反復の範囲(初期値、継続条件)	配列のシフト(追加・削除・逆順) 配列の加工(分割・併合・生成)

予備テスト3：整形出力や例外処理	
整形出力(桁数、区切、表形式)	例外処理(レアケース) 境界値検査

図 7 実行テスト系列における確認項目

3. 実行テスト系列

3.1. 予備テストと最終テスト

初心者の演習では, 必ずしも全員が問題を完答できるとは限らない．そこで，1 つの問題につき, 複数の予備テストおよび最終テストからなる実行テスト系列を提示する(図 4)．最終テストは, 題意を完全に満たす解答を要求するが, 予備テストでは部分的な仕様を満たせばよい(図 5)．図 6 は，不定個の整数値入力を受け取り，平均値を出力する問題である．この問題に対し，番兵による入力処理の確認，平均値計算の確認，出力書式の確認，などの予備テストを提示する．

予備テストでは, 最終テストへ至る前の段階的な確認を行う．予備テストの正解には，部分的な得点を与える．何度でも挑戦できるよう，失敗でも減点しない. 入出力のサンプルは公開されており，エラー情報も提示するため，積極的にテストに取り組み, その結果を元にデバッギングを進められる．

最終テストは, 十分な量の入出力サンプルを用いるが，学生には提示されず，結果のみを通知するブラックボックスとして行う. 正解には大きな得点を与えるが, 失敗すれば減点となる. 一般的なコンテストでは, この最終テストに該当するテストが用意されている. 最終テストで入出力サンプルを提示しない理由として, 具体的な処理を実装せず, 示したサンプルにのみ対応したようなコードを防ぐためである．また，予備テストを利用して慎重にテストを繰り返すことを促す効果も狙っている．

学生は，自身の理解度に応じ，最終テストの正答を目指して自由に実行テストを利用できる．プログラミングが不得意な学生には，完全な解答への着手の糸口となる. 中間目標として，段階的な実装を誘導する. 最終テストへの正答が難しいと判断した学生も，部分点を狙って，諦めずに挑める. プログラミングが得意な学生は，一気に最終テストの正答を狙ってもよい．ただし，最終テストに減点を設けることで，最後の確認として予備テストも利用させ, 慎重さを促す．

3.2. 包含系と加算系の実行テスト系列

中間目標としての予備テストは, 部分仕様の与え方と得点ルールで包含系と加算系の 2 種類を用意する. これらの実行テスト系列を学習項目や教育目的に応じて使い分け, 学生のプログラム作成を適切に誘導する．

包含系は，問題の仕様を緩めて，徐々に完成に近づけさせる. 図 6 中の入出力サンプルのように, パスすべきデータの難易度を段階的に高めていく. 副次的な処理を除々に追加し，完成度を高めていくような問題に適している. 学生に段階的なプログラム作成を意識させることができる．本研究でのコンテストでは，包含系を中心に予備テストの作成を行う．

加算系は，問題の仕様を分割して，別々の小さな問題として実装させ，最後に統合させる．機能ごとに部分的な動作を別の入出力サンプルを通して確認する. 関数ごとの実装を確認するような実行テスト系列では, 予備テストは単体テスト, 最終テストは結合テストに相当する．例えば，番兵式入力に関するコード，加算と除算に関するコードなどを別々に作成させ, 最終的に 1 つのコードに必要な処理を統合する. これは, 処理をモジュール単位に分割しやすい問

題に適している.

本研究のコンテストでは, 主に包含系の予備テストを採用する. これは, 加算系を実際にコンテストで利用する場合は, 基本的な処理は理解できているものとして出題するため, ある程度難易度が高く, 規模の大きな問題に適用することになる. そのため, 短時間で行う授業中のコンテストには, 余り適していない. また, 包含系では, 最終テストの入出力サンプルを, そのまま予備テストにも流用できるのに対し, 加算系では完全に別の入出力サンプルが必要となり, 運用面のコストが大きくなる. 以上の 2 つの点から, 包含系の予備テストを利用する.

3.3. 包含系の予備テストの詳細

包含系の予備テストでは, 入力範囲の限定と出力形式の許容の 2 通りを考える. それらを組み合わせて, 予備テストを構成する.

入力制限では, 仕様が要求する入力範囲内の一部でのみ正しく実行できればよい. テストケースとしては, 例外処理となるデータ, 継続条件や分岐条件での境界値を避ける. また, 予備テストで用いる入出力サンプル以外では, 正しく計算できなくてもよい. 本来の計算方法とは異なり, その入力でしか通用しない実装でも構わない. 例えば, 二次方程式の解法で判別式が正の場合のみ, 文字列処理で全てが英大文字の場合のみ, といったケースである.

出力許容では, 照合において, 一部のデータを無視して採点する. 具体的には, 仕様で要求される出力の個数, 書式や精度において, 照合の柔軟さを設定する. 例えば, データ列の平均と最大値を求める問題で, 平均だけの出力が合えばよいとする出力許容を行う. ここでは, 最大値の出力がなかったり, 誤った値が出力されていなかったりしても構わず, 予備テスト正答とする. 書式や精度においては, printf 文の出力書式による空白の個数や小数点表示などの相違を無視し, データとして一致すればよいとする採点を行う. 例えば, 平均値の問題ならば, 出力桁数を指定されていたり, 計算途中の小さな誤差を見逃すなどである.

3.4. 包含系の予備テストの構成

包含系の予備テストは, 図 7 のようなガイドラインに沿って構成し, 問題内容に即した処理の確認を行う. 予備テスト 1 では, 入力値をデータとして正しく扱えるかを主な確認項目とする. まずは, 問題に取り組むきっかけとなる部分として, 確実に解ける内容で実行テストを行う. 予備テスト 2 では, 主要な反復処理や配列操作を確認項目とする. 問題の中心となる処理において, 一部の機能のみを確認したり, 例外を含まない簡単な例で実行したりする. 予備テスト 3 では, 最終テストに至る最後の確認として, 整形出力や例外処理の確認を行う. 最終テストとほぼ同様の仕様とすることで, 予備テスト 3 を最終テスト前の確認として利用することができる.

4. 入出力サンプルと照合基準

4.1. 照合基準の概要

プログラムの自動判定を適正に行なうには, 問題文において, 処理内容だけでなく, 入力データの与え方, 出力の書式などの仕様も完全に指定しておく必要がある. 最終テストでは, 入力サンプルそれぞれに対し, 模範プログラムと解答プログラムの出力が文字列として完全に一致すれば, 完答と判定する. しかし, 予備テストの仕様に合わせて, 入出力サンプルを個々に用意したり, 予備テスト用に模範プログラムを修正するのでは, 問題作成のコストが大幅に増大する. そこで, 包含系の予備テストでは, 最終テストの入出力サンプルを流用し, 部分的な一致でも正答と判定する照合基準を設ける.

4.2. 行単位の単語単位での一致

入力範囲の限定では, 最終テストの入出力サンプルのうち, 特別なケースを除いた標準的なものを用いればよい. 出力形式の許容では, まず, 照合における完全一致の粒度を, 行単位と語句単位に分ける. 通常は, 単語単位での照合を行い, 出力順と単語単位での文字列として一致を行う. 行単位での照合では, 主に出力書式が指定されている場合に用いる. 利点として, 厳密な出力を評価することができる. 欠点としては, 空白の個数などの細かな点でも誤りと判定してしまう点である. 単語単位の指定は, 出力順が一致すればよく, 幅広い問題に利用できる. 利点として, 問題の形式にあわせた柔軟な採点が行えること. 欠点として, 書式などの評価が行えない点が挙げられる.

4.3. 前方からの一致個数の指定

さらに, 前方からの部分一致に用いる個数を指定する. ここで, 0 を指定すれば, 末尾までの完全一致とみなす. 例えば, 語句単位で一致数 2 であれば, 出力サンプル "12 3 45" に対し, 実行結果 "12　3" や " 12 3 6" は正答となる. これらは, 一致数 0(完全一致)であれば, 誤答と判定される. また, 行単位であれば, 書式の異なる " 12　3　45" も誤答となる. 前方からの一致個数指定の効果としては, 出力データの照合範囲の指定を行なうことで, プログラムの一部だけが記述できた状態を評価できる点である. 例えば, 最大値と平均値を出力する問題に対し, 予備テスト系列の中で, 最初は最大値のみを照合範囲とし, それ以降は平均値も確認する, といった使い方ができる.

4.4. 数値としての一致と誤差の許容

数値としての一致と誤差の許容について説明する. 実数を扱う問題に解答する際, 出力桁数が指定されている場合がある. 例えば, 正解としての出力に”5.50”が指定されていると, 厳密には”5.500000”の出力では誤答となってしまう. また, 問題の早い段階では, 出力が整数となる例を用いて, 動作確認を行なうこともある. この際も, ”2.00”が正解の場合, ”2”では誤答となる. これらに対応するため, 数値としての照合を導入する. これにより, 書式の差によって誤答と判定される可能性は少なくなる. また, 統計処理などで丸め誤差が発生する場合も考慮し, ある程度の揺らぎは許容する.

4.5. コメント行の許容

正誤判定において, もう 1 つの問題点は, "N=?" などのプロンプトや, デバッグ用の出力への対処である. 仕様の厳密な指定からいえば, これらは誤答とすべきであるが, 予備テストを最終解答への誘導とするならば, これらを許容することが望ましい. 本来, プロンプト出力は, ユーザフレンドリなプログラム作成において推奨すべきである. システムの試行運用においても, プロンプト出力を行なったため誤答とされたことを出題ミスと捉えた学生も多かった[3]. そこで, 特定の文字から始まるコメント行は, 照合において無視されるようにする.

4.6. 関連研究での手法

本研究の関連研究について触れる. プログラミング演習において, 入出力のマッチングによる採点を行う研究として, 田上[7]や松本[8], 石原[9]が挙げられる. Java 言語でも, 同様の取組みを横濱[10]らが行っている. これらは, 宿題のような教師に提出する課題に対し, システムを用いて採点し, 教師の負荷を軽減する. 教師が最終的な課題の採点を行うことを支援することが目的である. ある程度固定された, 少数の簡単な入出力を扱う問題への対応を行っている. これに対し, 本研究では, 学生同士や教師が途中経過を確認できる仕組みを用意し, 不定個のデータ列を中心とした, 柔軟な採点方法に対応する.

文献[8]では，少数の簡単な入出力の問題を想定し，プロンプト出力も許容するため，妥当な出力として要求される語句が出現していれば正答とみなしている．しかし,ある一桁の数字が出力サンプルのとき，"0 1 2 3 4 5 6 7 8 9" とすれば，必ず正答になってしまう．また，許容する出力パターンを正規表現などで記述する方法も考えられるが, 模範プログラムからの生成が難しい．本研究では，特に，不定個のデータや表形式での入出力も扱い, 予備テスト系列として入出力サンプルを統括するため, 上記の方法を用いている．

5. 判定結果の即時通知

5.1. 判定結果に応じた情報提示

プログラムの正誤判定の結果は, 学生に即時に通知される(図 8)．この段階に応じ，コンパイルメッセージ，入出力サンプルと実行結果，提出したコード自体も併せて提示される(図 9).

判定が不正提出の場合, 提出コードを表示し，学生に確認させる．静的エラーの場合は，コンパイルメッセージの確認や，ローカル PC での再コンパイルを促す．実行打切の場合は，変数などの初期化の確認などを促す. 実行時エラーの場合は，そこまでの実行結果を表示する．誤答の場合は，学生は出力結果を確認し，どの処理が正しく動作していないのかを推測する必要がある．正答の場合は，解答が順調に進んでいるため，次の問題やテストに誘導する．

5.2. コンパイルメッセージ

静的エラーの場合, サーバ側でのコンパイルメッセージを表示する. コンパイルに成功した場合は何も表示されない. エラーや警告のメッセージのみが，英語で表示される．本研究のコンテストでは, ソースコードはサーバ側に提出する前に，ローカル PC でコンパイル，実行を行っていることを想定している．そのため，コンパイルメッセージは余り表示されず, 英語であることも大きな問題はないと考えている．

ただし, 教師がコンパイルエラーを把握したい場合は, 学生に積極的にサーバ側でコンパイルを行わせるようにすることになる. そのためには, 現在は英語で出力されるコンパイルメッセージの日本語化や, コンパイルメッセージを詳細にするなどの工夫が必要となる．

5.3. 入出力サンプルと実行結果の表示

予備テストでコンパイルに成功した場合, 入出力サンプルと実行結果を表示する. 最終テストの場合は, 入出力を明示しないブラックボックステストとして行うため, 結果に関わらず表示しない．また，実行打切や実行時エラーと判定された場合は, 出力サンプルと実行結果の照合は行わず，参考として表示するのみである．

将来的には, 一定の水準まで完成したソースコードを判定するだけでなく, 学生のデバッグに積極的に関わることも考えられる. その場合, 正誤の表示だけでなく, 具体的に各実行結果のどこが間違っているかをシステム側で推定し，学生に示すことが必要になる．

5.4. 提出ソースコード

即時判定と同時に, 提出したソースコードの内容を表示する. 提出のミスなどを再確認できる．将来的には，Web ページ上で，軽微なミスを修正するエディタ機能も検討する．また，コンパイルエラーの表示と併せて, 学生のデバッグを支援することも考えられる．例えば，エラーや警告の指摘されている箇所を明確に表示したり，予想される原因などを示す，などである．ただし，余り多くのデバッグ情報を学生に提示すると, 学生が自分自身で考える機会を奪うことにも繋がるため, 現在ではソースコード表示自体への特別な支援は行っていない．

5.5. 個人と教室の進捗状況の表示

コンテスト中は，問題閲覧ページの上部に，

各問題の実行テストの進捗状況，得点と順位が速報として表示される．また，教室全体の順位表示，個人単位の提出履歴のページも用意する．順位表示ページでは，共時的な進捗確認として，教室全体の進捗状況を一覧する．他の学生の解答状況や得点が確認でき，全体の中での自分の位置付けを把握して，学生間の競争意欲を促進させる．提出履歴ページでは，通時的な進捗確認として，学生個人の提出履歴やテスト結果を掲載する．過去に提出したソースコードの内容を確認し，自分の弱点を把握したり，今後のコーディングの参考にする．

6. おわりに

大学情報系学科の入門的 C 授業において，初心者向けの小コンテスト形式でのプログラミング演習を提案し，大会運営の支援サーバ tProgrEss を開発した．tProgrEss は，サーバ側に提出された解答コードを自動採点し，コンテストの進捗状況を即時に表示する．中間目標としての実行テスト系列による正誤判定に対応するため，実行結果と出力サンプルとの柔軟な照合機能を実現した．また，コードの剽窃を検出する機能を組み込んだ．現在，初級 C 言語のプログラミング演習での運用中であり，その実践結果を分析して，機能の妥当性を検証する．

文　献

[1] 村井万寿夫，"学習意欲を高めるための手立てについて"，JSET　研究報告集，JET03-4，pp.31-36，(2003).

[2] 倉田英和，他，"実行テストを用いたコンテスト形式の入門的 C プログラミング演習の大会運営サーバの開発"，情処研報，Vol.2006，No.108，pp.9-16，(2006).

[3] 倉田英和，富永浩之，林敏浩，垂水浩幸，"実行テストによるプログラム判定を用いた初級 C プログラミング演習支援と授業実践"，情処研報，CE91，pp.11-18，(2007).

[4] 富永浩之，他，"コンテスト形式による初級Cプログラミングの演習支援"，情処研報，Vol.2008，No.42，pp. 49-56，(2008).

[5] 川崎慎一郎，富永浩之，"競争型学習を取り入れた入門的 C プログラミング演習 － 演習支援サーバ tProgrEss の出題解答と採点結果のページ表示の改良 －"，信学技報，Vol.109，No.335，pp.187-192，(2009).

[6] Ozy 著，やねうらお 編，"ShortCoding -職人達の技法-"，毎日コミュニケーションズ，(2007).

[7] 田上恒大，阿部公輝，"比較的大きなプログラミング課題のための指導採点システム"，情処研報，Vol.2006，No.16，pp.135-140，(2006).

[8] 松本真吾，酒井三四郎，"プログラミング学習支援のためのプログラム正誤判定システム"，JSiSE 第 32 回全国大会，pp.68-69，(2007).

[9] 石原俊，田口浩，島川博光，"多数の採点項目による C プログラミング実技試験の自動採点"，IPSJ 第 6 回 FIT 講演論文集，pp.393-394，(2007).

[10] 横濱彰則，海谷治彦，海尻賢二，"実行によるプログラムの診断 -Java ReflectionAPI によるプログラムのテスト-"，信学技報，Vol.104，No.725，pp.37-42，(2005).

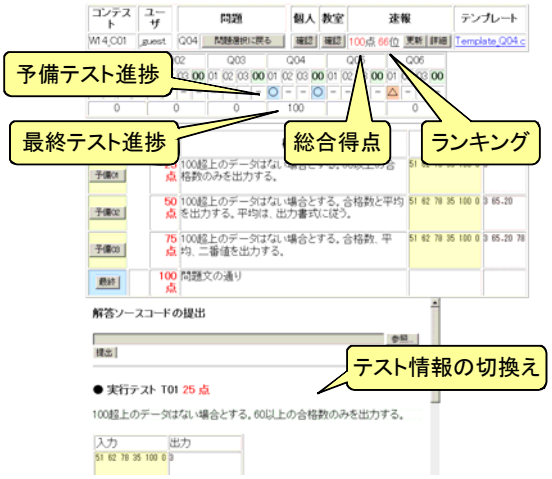

図 8 問題閲覧ページの出題解答モード

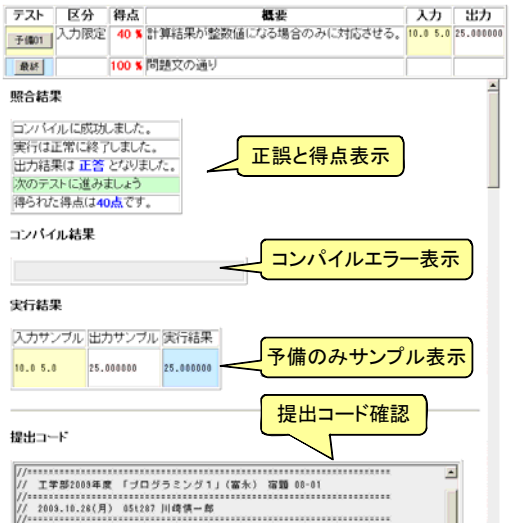

図 9 問題閲覧ページの採点結果モード