マークアップによるC言語プログラミング試験採点システム

石原　俊† 　田口　浩†† 　高田　秀志††† 　島川　博光†††

† 立命館大学理工学部情報学科　〒 525-8577　滋賀県草津市野路東
†† 立命館大学理工学研究科　〒 525-8577　滋賀県草津市野路東
††† 立命館大学情報理工学部　〒 525-8577　滋賀県草津市野路東
E-mail: †{shun,hiro}@de.is.ritsumei.ac.jp, ††{htakada,simakawa}@cs.ritsumei.ac.jp

あらまし　ほとんどの情報系大学が学生にプログラミング試験を課している．受験者が多いため，時間的制約により教員はシステムによる自動採点を採用する．システムによる採点は多くの採点基準を設けにくく，あまり細やかではない．そこで本論文では，教員の模範解答プログラムと，解答プログラムの動作確認用 main 関数の 2 つから採点システムを生成する手法を提案する．教員は main 関数に，多様な入力データと学生に示されるコメントを，特殊なマークアップ言語を用いて記述する．システムはマークアップされた関数を解析し，入力データの数だけテストドライバを生成する．さらに，生成されたすべてのテストドライバを用いて，多様な入力データによる採点を行う．本論文では提案手法をシステムとして実装し，実験による性能評価を行った．

キーワード　プログラミング実技試験　自動採点　ブラックボックステスト　マークアップ言語

Automated Marking System for C Programming with Mark Up Tags

Shun ISHIHARA†, Hiroshi TAGUCHI††, Hideyuki TAKADA†††, and Hiromitsu SHIMAKAWA†††

† School of Informatics, Ritsumeikan University　Noji-higashi, Kusatsu-shi, Shiga, 525-8577, Japan
†† Graduate School of Science and Engineering, Ritsumeikan University　Noji-higashi, Kusatsu-shi, Shiga, 525-8577, Japan
††† Department of Information Science and Engineering, Ritsumeikan University　Noji-higashi, Kusatsu-shi, Shiga, 525-8577, Japan
E-mail: †{shun,hiro}@de.is.ritsumei.ac.jp, ††{htakada,simakawa}@cs.ritsumei.ac.jp

Abstract　Most of universities which have information science department conduct programming skills tests. Teachers are apt to use automated marking systems, because they must mark extremely many programs. However, marking with those systems is hard to provide many marking criteria, and it is not precise. This paper suggests a method which generates marking systems from a model solution program, a students solution program, and a special main function. Using a mark-up language, teachers embed diverse input data and comment in main function. The system analyzes the marked-up main solution, and creates test driver programs. Moreover, the system calculates scores for diverse input data, using all test driver programs. The ability of the proposed method is evaluated with an implemented system.

Key words　Programming Skills Test, Automated Marking, Black Box Test, Mark Up Language

1. はじめに

現在，ほとんどの大学の情報系学部において，C 言語プログラミング実技試験が実施されている．学生数の多い大学では，採点のさい，時間的制約から教員がすべての解答を目で見て採点することは非常に難しい．そのような場合，文献 [2] [7] に挙げられるように，教員は自動採点システムを作成する．自動採点システムは，回帰テスト [5] のように, 学生の解答と模範解答とを同じ入力データを用いて実行し，得られる結果の比較によって採点を行う．自動採点システムを用いることで，教員は膨大な数の解答を，短時間で採点することができる．反面，自動採点システムによる採点は，入力データ，すなわち採点項目を多く設けられず，その採点基準はあまり細やかではない．また，採点項目を多く設けられるような自動採点システムは，構造的に複雑であり，作成の負荷が大きい．プログラミング課題を細やかに採点することを支援する手法には，動的解析手法 [3] や，静的解析手法 [4] などがある．両手法とも，重点は，学生へ詳細なフィードバックを返すことに置かれている．そのため,

出題可能な問題形式への制限や，システムのための教員の作成物の多さなどから，実技試験の採点には適さない．重点が教員の負荷軽減に置かれた手法には，ほかに GAME [6] [8] がある．GAME が学生に開示する採点結果は，数値的な点数や，エラーメッセージなどの定量的なもののみである．そのため，学生は，採点結果をから，自分のプログラムのどこに欠陥があるのかを確認することができない．

そこで，本論文では，模範解答プログラムと，それのテストドライバとなる main 関数のみを用いて，多様な入力データでの自動採点を行う Mark Up Marking 手法を提案する．教員は main 関数に，入力データや学生に示されるコメントからなる採点情報を，特殊なマークアップ言語を用いて記述し，模範解答プログラムとともにシステムに入力する．システムはマークアップされた main 関数を解析し，入力データごとに，その入力データでのテストドライバを生成する．次に，生成されたすべてのテストドライバを用いて，テストドライバごとに自動採点システムを構成する．最後に，構成されたすべての自動採点システムを逐次起動し，採点結果の集計を行う．集計された結果は，教員には全学生の採点結果が，学生には自分の採点結果のみが返却される．これにより，教員には負荷軽減の効果が，学生には従来より詳細なフィードバックがもたらされる．

本論文では，多様な入力を用いた採点の所要時間が，Mark Up Marking 手法によってどの程度削減されるのかを検証するため，実験を行った．その結果，採点における所要時間は，手動採点に比べ，2 分の 1 にまで削減された．

2. 教員負荷軽減のための自動採点

2.1 プログラミング実技試験

現在，ほとんどの大学の情報系学部において，C 言語プログラミング実技試験が実施されている．試験で用いられる問題の多くは，書式や機能が指定された関数を学生に作らせるような形式で実施されている．このような試験問題の例を図 1 に示す．教員は，図 1 のような試験問題の他に，学生が使用する動作確認用 main 関数を用意する．動作確認用 main 関数の例を図 2 に示す．動作確認用 main 関数とは，試験中に学生が自分の作った関数の動作を確認するための関数である．動作確認用 main 関数は大まかに以下の 3 種類の部分から成り立っている．

(1) 動作確認対象となる関数の引数へ何らかの値を設定する箇所

(2) (1) で設定した値を与えて関数を呼び出し，戻り値を保存する箇所

(3) (2) で保存した戻り値の内容を出力する箇所

図 2 の例では，5 行目と 6 行目が (1) に該当し，8 行目から 13 行目が (2)(3) に該当する．学生は，動作確認用 main 関数と，自分の作成した関数をコンパイル結合して実行することで，あらかじめ与えられた入力データについて，関数の動作を確認することができる．多様な入力データを用いて動作確認を行う場合，学生は (1) の部分を様々なデータで置き換える必要がある．

以下に示す仕様に従い，関数getWordを作りなさい．

●機能:
英文テキストsentence**中で最初に出現する単語を**word**に格納する．**
ただし，sentence**中に一つも単語が存在しない場合は**wordにNULL**を格納する．**
ここで言う単語とは，スペース以外の文字で始まり，
スペースおよびヌル文字が出てくるまでの文字の列とする．
また単語に含まれるアルファベットは全て小文字に変換してからword**に格納する．**
すなわち，"That"**は**"that"**として格納する．**
最後にsentence**中で検出した単語の次の位置へのポインタを返す．**
以下に入出力の例を示す．

（例1） sentence=" " **の場合** word=NULL**となり** sentence**の末尾のアドレスを返す**
（例2） sentence=" dog cat bird" **の場合** word="dog" **となり**
dog**と**cat**の間にあるスペースの位置へのポインタを返す**
（例3） sentence="FOX " **の場合** word="fox" **となり**
fox**の後にあるスペースの位置へのポインタを返す**

●書式:
char* getWord (char *sentence, char *word)

●引数:
sentence – **分析対象となる英文テキスト**
word – **最初に出現する単語を格納するための文字列**

●戻り値:
sentence**中で検出した単語の次の位置のポインタ**

図 1　試験問題の例

```
1  main()
2  {
3     char  name1[100], name2[100];
4
5     strcpy(name1, "tamura" );
6     strcpy(name2,  "kimura“ );
7
8     if ( preceding(name1, name2 ) == 1 )
9        printf("%s is earlier than %s¥n", name1, name2 );
10    else if ( preceding(name1, name2 ) == -1 )
11       printf("%s is earlier than %s¥n", name2, name1 );
12    else
13       printf("%s is equal to %s¥n", name1, name2 );
14 }
```

図 2　動作確認用プログラムの例

2.2 自動採点システム

学生数の多い大学において，プログラミング実技試験の採点は，教員にとって多大な負荷である．図 1 のように，書式や機能が指定された関数を作らせる形式の問題は，システムによる採点の自動化が可能である．そのため，多くの教員は図 1 のような形式の問題を作成し，採点を自動化する．システムによる自動採点の流れを図 3 に示す．まず，システムは学生の解答と

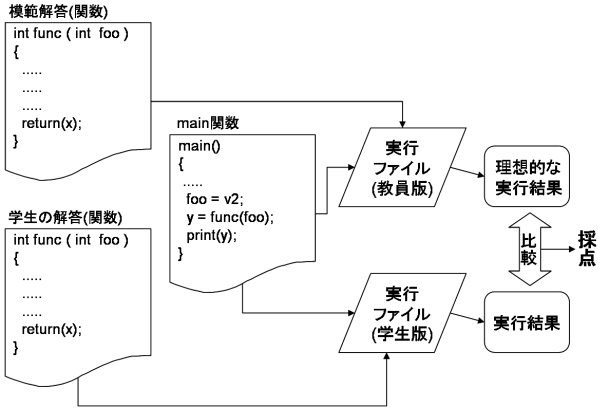

図 3　システムによる自動採点の流れ

教員の模範解答を，それぞれ main 関数とコンパイル結合し，実行形式ファイルを作成する．コンパイル結合に用いる main 関数は，動作確認用 main 関数と同様の構造である．次に，システムは，学生版実行形式ファイルと教員版実行形式ファイルの両者を実行し，出力結果を保存する．保存された 2 つの出力結果のうち，教員版実行形式ファイルからの出力結果を，理想的な実行結果であると仮定する．最後に，学生版実行形式ファイルからの出力結果と，教員版実行形式ファイルからの出力結果との比較を行う．比較の結果，両者が一致していれば学生の解答は正解と判断され，一致していなければ不正解と判断される．本論文では，このような流れによる採点を，複数の学生に対して自動で行うようなシステムを，自動採点システムと定義する．

自動採点システムの問題点として，評価が細やかではないという点が挙げられる．この問題点は，採点時に用いる採点項目の少なさに起因する．自動採点システムにおける採点項目は，main 関数内で採点対象となる関数に与えられる入力データに対応する．負荷の軽減に主眼の置かれた自動採点では，教員は単一の入力データのみを用いた採点を採用する．そのため，0 点か 100 点のどちらかでしか学生を評価できず，学生の学習意欲を削ぎかねない．

評価の細かさに主眼の置かれた自動評価支援手法には，動的解析手法 [3] や，静的解析手法 [4] などがある．これらの手法における評価内容は，ソースコードの質や，問題の解決に必要な知識分野の提案などを含んでおり，学習意欲の向上には効果的である．反面，教員はシステムに対して多くの入力物件や手動作業を要求されるため，あまり負荷を減らすことができない．また，評価の詳細さと引き換えに，学生は課題の提出後，評価を即座に受けることができず，学習意欲を削がれてしまう．自動採点と既存手法の両方の問題点を解決するには，以下の 3 点のバランスを考慮した手法が必要である．

- 評価の細やかさ
- システムの利用に必要な教員の負荷
- 学生の採点結果へのアクセス性

2.3 動的解析手法

プログラミング課題の評価を支援する手法の 1 つに，動的解析手法がある．動的解析手法において，問題は，何箇所かが欠落したプログラムを学生に与え，欠落した箇所のコードを書かせるような形式で与えられる．欠落した箇所にどのようなコードを書くべきかは，問題において指示される．システムは，このような形式の問題に対する学生の解答プログラムを解析し，評価を行う．評価の結果，学生の解答プログラムに間違いが発見された場合，間違いの理由や改善へのアドバイスなどのフィードバックが学生へ返される．これにより，教員は課題評価の負荷が軽減され，学生は課題に対する迅速な指導が受けられるようになる．

動的解析手法には，2 つの問題点がある．1 つ目の問題点は，詳細なフィードバックと引き換えに，出題可能な問題の形式が限定されることである．問題の形式が限定されると，教員は学生に対して多様な演習を課すことができない．2 つ目の問題点は，教員の用意しなければならないものが多いことである．動的解析手法では，問題の他にアダプターコードと呼ばれる特殊なコードを問題ごとに用意する必要がある．また，学生へのフィードバックが細やかであるため，多様なフィードバックコメントも用意しておかなければならない．

2.4 GAME

プログラミング課題の評価を支援するシステムのひとつに，GAME がある．GAME において，問題は，複数の関数と，その中から呼び出す関数を指定するためのコマンドラインメニュー機能を持つ main 関数を作らせるような形式で与えられる．教員は，コマンドラインメニューを操作するための指示リスト，入力値リスト，および入力に対応した出力値リストの 3 つをシステムに入力する．GAME は，これら 3 つの入力ファイルを解析し，学生のプログラムの評価を行う．そのさい，入力値と出力値の対応による採点以外に，ソースコード内のコメント量や，インデンションについても評価を行う．これにより，学生は自分のプログラムについて，動作の正確さとプログラミングスタイルの両方面からの評価を受けることができる．

GAME には，2 つの問題点がある．第一の問題点は，プログラムの動作の正確さについての評価が十分ではない点である．GAME が学生に開示する採点結果には，採点に用いた入力データ数と，その中で正解と判断された入力データ数しか含まれない．したがって，学生は，自分のプログラムが，どのような入力を正しく処理できないのかを確認できない．第二の問題点は，出題可能な問題の形式が，実技試験で用いるのに適さないことである．GAME が指定する問題形式において，学生は問題として問われている関数の他に，それらを呼び出すコマンドラインメニューを作る必要がある．学生は，コマンドラインメニューを完成させない限り，他の関数を完成させていても正当な評価を得られない．このような仕様は，時間制限のある実技試験において，正当な評価の障害となる．

3. Mark Up Marking 手法

本論文では，多様な入力による採点を容易に実現するため，Mark Up Marking 手法を提案する．提案手法をシステムとして実装したものを Mark Up Marker と定義する．main 関数を用いて複数の入力データを供給する方法として，コマンドライン引数を用いる方法と，main 関数自体を複数用意する方法の 2 つがある．本論文で提案する Mark Up Marking 手法は，後者の方法で複数の入力データを供給する．

3.1 情報の流れ

Mark Up Marking 手法の概略図を図 4 に示す．最初のステップとして，教員と学生の両者は，Mark Up Marker に対して，プログラムを記述したファイルを入力する．学生が入力するファイルは，自分の作成した解答プログラムである．教員が入力するファイルは，模範解答プログラムと，採点用 main 関数という特殊な main 関数である．採点用 main 関数とは，動作確認用 main 関数に独自のタグを用いて複数の採点情報を追記したものである．採点情報は，以下の 3 つの情報から構成される．

- 学生の解答に適用する入力データ
- 入力データに対する配点

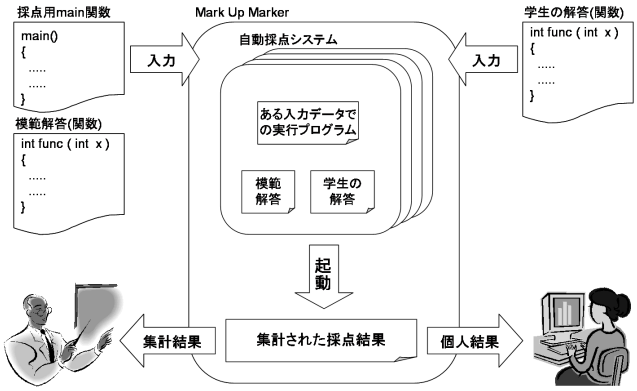

図 4　Mark Up Marking 手法の概略図

- 学生に提示するコメント

次のステップとして，Mark Up Marker は採点用 main 関数を解析し，採点情報内の数だけ，一組の採点情報に対応した main 関数を示したプログラムを生成する．さらに，生成されたすべてのプログラムと，双方の解答を用い，採点情報の数だけ自動採点システムを構成する．最後のステップとして，システムは構成されたすべての自動採点システムを逐次起動し，結果の集計を行う．集計された結果は，学生には個人結果として，教員には全体結果としてそれぞれ通知される．

3.2　3 種類のタグ

2.1. で説明したように，受検者は動作確認用 main 関数の，実引数設定箇所を多様なコードで差し替えれば，多様な入力データによる動作確認を行うことができる．Mark Up Marker は，この差し替え作業を自動化することで，多様な入力データによる採点を実現する．本論文では，差し替え作業を自動化するために，独自のマークアップ言語を定義する．マークアップ言語は，以下の 3 種類のタグから構成される．

args タグ　実引数設定箇所を指定するためのタグ

replace タグ　指定された実引数設定箇所に対して差し替え元となるコードを記述するためのタグ

case タグ　差し替えコードの組み合わせなどから構成される，採点項目を記述するためのタグ

これらのタグは，C 言語コンパイラからはプログラムコメントに見えるように，/*と*/で囲まれている．図 5 にタグ同士の関連を示す．args タグは，実引数設定箇所に目印をつけておく目的で用いられる．replace タグは，args タグで目印が付けられた箇所に対して，差し替えの候補となるコードを記述する目的で用いられる．多様な入力データによる採点が目的であるため，一般的に差し替えの候補となるコードは複数存在する．また，実引数設定箇所も，動作確認対象関数により複数存在する．このような場合，入力データを構成するには，差し替えコードの組み合わせ情報が必要である．case タグは，無数に存在する差し替えコードの，どれとどれを組み合わせて関数への入力データとするのかを指定するために用いる．組み合わせ情報の他に，case タグ内には，配点やコメントなどの情報が含まれる．教員は，これら 3 種類のタグを，動作確認用 main 関数に，コメントとして追記する．

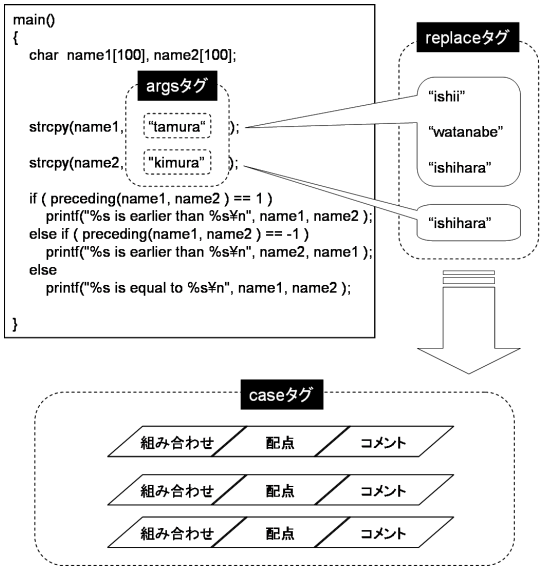

図 5　タグ同士の関連

3.3　教員によるマークアップ

3 種類のタグにより採点情報が追記されたプログラムの例を図 6 に示す．まず，教員は，動作確認用 main 関数内の実引数設定箇所を args タグで括る．args タグは以下のような書式で表現される．

$$< args\_n > \cdots < /args\_n > (n \geqq 1)$$

n は args タグの識別番号である．実引数設定箇所が複数存在する場合，教員は args_1，args_2…と識別番号を増やしながらすべての箇所を括っていく．図 6 の例では，実引数設定箇所として，args_1 と args_2 の 2 箇所が括られている．

次に，教員は，args タグを用いて指定した実引数設定箇所に対する差し替えコードを，replace タグを用いて指定する replace タグは以下のような書式で表現される．

$$< replace\_n\_m > \cdots < /replace\_n\_m > (n, m \geqq 1)$$

n は args タグに振られた識別番号に対応している．教員は，n

```
main()
{
   char  name1[100], name2[100];

   strcpy(name1, /* <args_1> */ "tamura" /* </args_1> */  );
   strcpy(name2, /* <args_2> */ "kimura" /* </args_2> */  );

   if ( preceding(name1, name2 ) == 1 )
      printf("%s is earlier than %s¥n", name1, name2 );
   else if ( preceding(name1, name2 ) == -1 )
      printf("%s is earlier than %s¥n", name2, name1 );
   else
      printf("%s is equal to %s¥n", name1, name2 );

   /* <replace_1_1> "ishii" </replace_1_1> */
   /* <replace_1_2> "watanabe" </replace_1_2> */
   /* <replace_1_3> "ishihara" </replace_1_3> */
   /* <replace_2_1> "ishihara" </replace_2_1> */

   /* <case> 1+1, 30, 簡単に分からない場合を見分けられる </case> */
   /* <case> 2+1, 10, 簡単に分かる場合を見分けられる </case> */
   /* <case> 3+1, 20, 完全に一致する場合を見分けられる </case> */
}
```

図 6　採点情報が追記されたプログラムの例

によって，どの実引数設定箇所に対する差し替えコードかを指定する．*m* は replace タグの識別番号である．n 番目の実引数設定箇所に対して差し替えコードが複数存在する場合，教員は replace_*n*_1，replace_*n*_2…と識別番号を増やしながらコードを指定していく．例えば replace_2_3 は，args_2 で指定された箇所に対する 3 番目の差し替えコードを意味する．図 6 の例では，args_1 に対する差し替えコードとして replace_1_1，replace_1_2，replace_1_3 が，args_2 に対する差し替えコードとして replace_2_1 が，それぞれ指定されている．

最後に，教員は，case タグを用いて採点項目を記述する．case タグは以下のような書式で表現される．

$$< case > \text{組み合わせ,配点,コメント} < /case >$$

採点項目は，コンマで区切られた 3 つの情報から構成される．第一の情報は，差し替えコードの組み合わせである．args タグで囲まれた実引数設定箇所が j 個あるとき，組み合わせは以下のような形式で表現される．

$$m_1 + m_2 + m_3 + \cdots + m_i + m_j$$

m_i は args_*i* で括られた実引数設定箇所に対する *m* 番目の差し替えコード，すなわち replace_*i*_*m* で括られたコードを意味する．第二の情報は，採点項目に与えられる配点である．第三の情報は，採点項目に対する，教員のコメントである．教員は，採点項目の作成意図などをコメントに記述する．採点結果の返却時に，学生はコメントを見ることで，自分のプログラムの欠陥を確認することができる．図 6 の例では，3 つの case タグで囲まれた採点項目が存在する．1 番目の case タグ内の情報は，replace_1_1 と replace_2_1 の組み合わせに相当し，この入力データに対し正しい出力が得られた場合，30 点が与えられることが示されている．以下同様に，2 番目の case タグ内の情報には，replace_1_2 と replace_2_1 の組み合わせ，3 番目の case タグ内の情報には，replace_1_3 と replace_2_1 の組み合わせが記述されている．

4. Mark Up Marker

Mark Up Marker は，教員用システムと学生用システムの 2 つから構成される．教員用，学生用ともに，Web ベースのシステムである．

4.1 試験の作成

教員用システム，学生用システムの画面を図 7，図 8 に示す最初に，教員は，図 7 の教員用システムの試験作成フォームより，試験を作成する．そのさい，試験名，採点用 main 関数，模範解答の 3 つを指定する．試験が作成されると，教員用システムの試験表示領域に，試験名とタイムスタンプが表示される．さらに，図 8 の学生用システムの試験表示領域にも，同様の情報が表示される．この状態で，学生は，試験に対して解答を提出することが可能となる．図 7，図 8 の例では exam1，exam2，exam3 の 3 つの試験が作成されている．

4.2 試験に対する操作

学生の解答がすべて提出されると，教員は，現在提出されて

図 7　教員用システム

図 8　学生用システム

いる解答に対して採点を行う．採点には，試験作成時に提出された採点用 main 関数と模範解答が用いられる．採点が終了すると，図 7 の試験名に採点結果へのリンクが張られる．教員は，リンクをクリックすることで，採点結果を確認できる．実際に表示される採点結果の画面を，図 9 に示す．図 9 の例では，exam3 の採点結果が表示されている．

5. 実験による評価

本論文では，多様な入力を用いた採点の所要時間が，本システムによってどの程度削減されるのかを検証するため，実験を行った．

5.1 解答プログラムの収集

実験の前段階として，解答プログラム収集を目的としたプログラミング実技試験を実施した．試験問題は，図 1 と同様のものを用いた．図 1 の問題を，立命館大学情報理工学部情報システム学科の 3 回生 12 人に，90 分の制限時間内で解かせた．試験時には，図 1 の問題の他に，予め作成しておいた動作確認用 main 関数も提供した．学生が試験中に参照可能な物件は，C 言語の参考書および解説 Web ページのみに限定した．

5.2 実験と結果

最初に，妥当な採点項目の作成を行った．採点項目の作成のために，実技試験で得られた 12 人分の解答プログラムの中から，1/3 にあたる 4 人分のプログラムを無作為に選択した．さらに，選択したプログラムを実際に目で見ながら，点数に差の

採点結果
(exam3)

アカウント	コメント	配点	得点
cs009046	wordにNULLを入れる処理ができている	15	15
	"fox_cat_dog"のような単純なケースをさばける	10	10
	アルファベット以外の文字から成る文字列をさばける	5	5
	文字列の終端がスペースではなくヌル文字である文字列をさばける	20	20
	wordにNULLを入れる処理ができており、戻り値が正しい	15	0
	"fox_cat_dog"のような単純なケースをさばけ、戻り値が正しい	10	10
	アルファベット以外の文字から成る文字列をさばけ、戻り値が正しい	5	5
	文字列の終端がスペースではなくヌル文字である文字列をさばけ、戻り値が正しい	20	0
	計	100	65
cs010046	wordにNULLを入れる処理ができている	15	15
	"fox_cat_dog"のような単純なケースをさばける	10	10
	アルファベット以外の文字から成る文字列をさばける	5	5
	文字列の終端がスペースではなくヌル文字である文字列をさばける	20	20
	wordにNULLを入れる処理ができており、戻り値が正しい	15	15
	"fox_cat_dog"のような単純なケースをさばけ、戻り値が正しい	10	10
	アルファベット以外の文字から成る文字列をさばけ、戻り値が正しい	5	5
	文字列の終端がスペースではなくヌル文字である文字列をさばけ、戻り値が正しい	20	20
	計	100	100

図 9　採点結果の画面

つく採点項目を作成した．そのさい，採点項目の作成には 39 分 6 秒の時間を要した．作成した 6 つの採点項目を表 1 に示す．表 1 中の採点項目のうち，(1)〜(3) は実行結果の出力のみを，(4)〜(6) は実行結果の出力と戻り値を調べている．

次に，採点項目の作成に用いた 4 人分の解答を除く，残る 8 人分の解答を採点項目ごとに，2 通りの方法で採点した．第一の方法は，Mark Up Marker を用いた採点である．Mark Up Marker を用いた採点とは，教員が作成済みの動作確認用 main 関数に採点情報をマークアップし，採点用 main 関数を作成する過程のみを指す．Mark Up Marker を用いた採点に要した時間は，12 分 39 秒であった．第二の方法は，手動採点である．具体的に

表 1　採 点 項 目

項目番号	入力データ例	配点	コメント
(1)	”　”	15	Word に NULL を入れる処理ができている
(2)	”fox cat dog”	10	もっとも単純なケースを処理できる
(3)	”fox”	20	単語の終端がスペースではなくヌル文字である文字列を処理できる
(4)	”　”	15	(1) の条件を満たしており，かつ戻り値が正しい
(5)	”fox cat dog”	10	(2) の条件を満たしており，かつ戻り値が正しい
(6)	”fox”	20	(3) の条件を満たしており，かつ戻り値が正しい

表 2　手動採点の所要時間

項目番号	所要時間
(1)	22 分 19 秒
(2)	14 分 49 秒
(3)	08 分 21 秒
(4)	09 分 41 秒
(5)	11 分 00 秒
(6)	06 分 49 秒

は，適当なテキストエディタで解答プログラムを開き，採点項目の入力データを処理できるかどうかを目で確認していく過程を指す．そのさい，コンパイラやドキュメントなど，PC 上で一般的に使用できる物件は使用可とした．手動採点に要した時間は表 2 のとおりである．

5.3　考　　察

実験結果をグラフ化したものを，図 10 に示す．採点部分についての所要時間は，Mark Up Marker によるものが 12 分 30 秒，手動採点が全項目合計で 72 分 59 秒である．したがって，採点部分については，Mark Up Marker によって，教員の採点に要する時間的負荷が従来のおよそ 6 分の 1 に削減されたことが分かる．採点工程全体としては，両手法で共通に，採点項目の作成に要する時間が含まれる．採点項目の作成に要する時間は，39

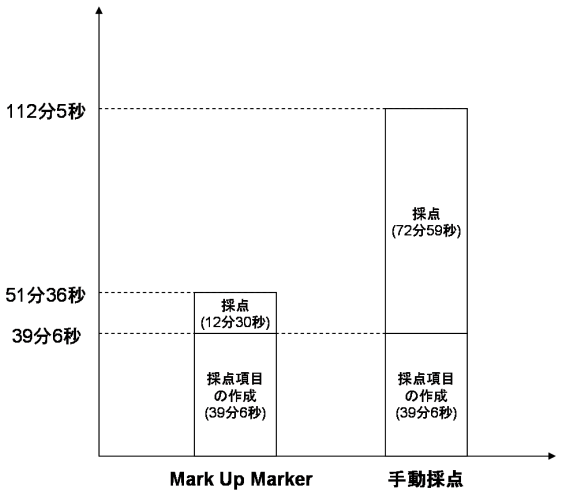

図 10　実験結果のグラフ

分 6 秒である．したがって，全体としては，Mark Up Marker によって，教員の採点に要する時間的負荷が従来のおよそ 2 分の 1 に削減されたことが分かる．また，手動採点では，採点項目の総数 48 個 (6 項目を 8 人分) のうち 9 個の項目について採点ミスが見られた．以上より，Mark Up Marker は手動採点に比べ，多様な入力データによる採点を，正確かつ容易に実行できるということが言える．

5.4　動的解析手法との比較

本論文で提案した Mark Up Marking 手法を，動的解析手法 [3] と比較する．動的解析手法には，出題形式が限定されることと，試験準備に対する教員負荷が高いという問題点がある．特に後者において，学生の意欲向上に有効なフィードバックを充実させるためには，教員負荷は大きくなる．一方，Mark Up Marking 手法は，問題に対する制限が少ないため，学生に多様な形式の問題を課すことができる．さらに，Mark Up Marking 手法は，問題と，動作確認用 main 関数に採点情報を追記した関数を用意するだけで，評価と簡易的なフィードバックを実現することができる．

5.5　GAME との比較

本論文で提案した Mark Up Marking 手法を，GAME [6] [8] と比較する．GAME には，出題形式が試験に適さないことと，プログラムの動作の正確さについての評価が不十分であるという 2 つの問題点がある．前者の問題について，Mark Up Marking 手法では，学生は問題として問われている関数のみを作ればよい．採点時により正当な評価が可能であるという点で，Mark Up Marking 手法が定める形式は，実技試験に適している．後者の問題について，Mark Up Marker は採点結果に，プログラムの処理できない入力データの特性を含めている．これにより，学生は点数以外に自分のプログラムの機能的欠陥を確認することができ，学習意欲を向上させることができる．

5.6　技能を検証するフェーズ

大学におけるプログラミング教育は，2 つのフェーズに大別される．第一のフェーズは，講義や演習などの，学生のプログラミング技能を養成するフェーズである．技能を養成するフェーズにおける教員の目標は，プログラミングに必要な知識を学生に習得させることである．そのため，演習のさい，教員は提出された解答に対して，間違いの解決に必要な知識などの詳細なフィードバックを返す必要がある．詳細なフィードバックを返すことに特化した手法としては，[1] [3] [4] などが存在する．これらの手法の利用には，教員が個別に手動で行わなければならない作業や，システムに対する多くの作成物が必要である．

第二のフェーズは，実技試験などの，学生のプログラミング技能を検証するフェーズである．技能を検証するフェーズにおいて重要なことは，教員は学生の，学生は自分の，到達度を確認することである．到達度を確認することの主な目的は，学生に素早く得点を通知して，学習意欲を向上させることにある．よって，教員は，多数の解答に対する多数の採点項目での採点を，短時間で実施できることが望ましい．本論文で提案する Mark Up Marking 手法は，このような技能を検証するフェーズにおける採点の支援に特化した手法である．

6.　おわりに

本論文では，C 言語プログラミング実技試験を対象として，多様な採点項目による自動採点を，従来より低負荷で可能にするため，Mark Up Marking 手法を提案した．また，Mark Up Marking 手法を，Mark Up Marker として実装した．Mark Up Marking 手法では，教員は採点に必要な情報を，採点対象関数のテストドライバとなる main 関数に特殊なタグを用いて記述する．記述する情報は，採点に用いる採点項目，すなわち関数への入力データと，採点項目に対応する配点やフィードバックコメントから構成される．教員は，フィードバックコメントに，採点項目の作成意図などの情報を含める．Mark Up Marker は，main 関数内にマークアップされた情報に従い，採点および結果の集計を行う．集計された採点結果は，学生ごとにすべての採点項目について，正解か否かという情報と，採点項目に対応したフィードバックコメントを保持する．これにより，教員は学生の，学生は自分の，到達度を確認することができる．

また，Mark Up Marker が，多様な入力を用いた採点の所要時間をどの程度削減できるのかを検証するため，性能評価を行った．その結果，Mark Up Marker は，手動採点に比べ，採点に要する時間を 2 分の 1 にまで減少させた．

今後の課題として，マークアップ言語の改良，マークアップ言語の記述支援などを考えている．

文　献

[1] 小柳順一, 島川博光: ”コード・イディオムの構造解析を用いたソースコードの自動評価 ”, 第 5 回情報科学技術フォーラム, pp321-322 (2006)

[2] 内藤広志: “ プログラミング演習の総合支援システムの概要 ”, 第 3 回情報科学技術フォーラム, pp305-306 (2004)

[3] Nghi Truong, Paul Roe, Peter Bancroft: “ Automated Feedback for “ Fill in the Gap ”Programming Exercises ”, Australasian Computing Education Conference 2005, pp117-126 (2005)

[4] Nghi Truong, Paul Roe, Peter Bancroft: “ Static Analysis of Students ’Java Programs ”, Australasian Computing Education Conference 2004, pp317-325 (2004)

[5] Jiang Zheng, Brian Robinson, Laurie Williams, Karen Smiley: ”Applying Regression Test Selection for COTS-based Applications ”,

ICSE ’06, pp512-521 (2006)
[6] Michael Blumenstein, Steven Green, Ann Nguyen, Vallipuram Muthukkumarasamy: ”An Experimental Analysis of GAME: A Generic Automated Marking Environment ”, ITiCSE ’04, pp67-71 (2004)
[7] Lauri Malmi, Ari Korhonen, Riku Saikkonen: ”Experiences in Automatic Assessment on Mass Courses and Issues for Designing Virtual Courses ”, ITiCSE ’02, pp55-59 (2002)
[8] Michael Blumenstein, Steven Green, Ann Nguyen and Vallipuram Muthukkumarasamy: ”GAME: A Generic Automated Marking Environment for Programming Assessment ”, ITCC ’04 (2004)