プログラミングスタイルの診断システムの構築

関本 理佳　　海尻 賢二

信州大学 工学部 情報工学科
E-mail: rika@cs.shinshu-u.ac.jp

あ ら ま し　プログラミング教育において、プログラミングスタイルを意識させることは、良いプログラマを育てるために重要な要素である。我々は、入力されたCのソースプログラム中からプログラミングスタイルに違反する箇所を検出することにより、良形度の高いプログラムの作成を支援する、プログラミングスタイルの診断システムを試作した。本システムは、以下の特徴を持つ。(1) 個々のガイドラインをプログラムパターンとして記述し検査するので、検査項目の追加が容易に行なえ、拡張性に優れている。(2) 文脈に依存した項目や関数にまたがる項目など、単純なテキスト上のマッチングでは検索が難しいパターンの認識も可能である。我々は、文献等に載っている種々の悪形パターンを記述し、テキストユーティリティのプログラム群を対象とした認識実験を行ない、標準的なガイドラインが本システムにより正しく検出可能であることを確認した。さらに、初心者プログラマを対象としたスタイル検査実験により、本システムが悪形パターンの検査支援に有効に働き、本システムの利用により検査能力が向上することを実証した。

キーワード　プログラミングスタイル, プログラムパターン, プログラム認識, 診断

A diagnosis system of programming style

Rika SEKIMOTO and Kenji KAIJIRI

Department of Information Engineering,
Faculty of Engineering, Shinshu University

Abstract　Programming style plays an important role in program education for novice programmers. By obeying it, programs become readable and understandable. We aim at developing a support system for checking programming style. This system detects ill-formed patterns in a program and makes a diagnosis on programming style. This system has the following features; 1) We decided to use program patterns as description form of programming style, so it becomes easy to add the checking criteria, 2) This system can detect various patterns, for example, context dependent patterns and dispersed patterns extending two or more functions. It is difficult to detect these patterns by character based pattern matching. We checked that detection of various ill-formed patterns is possible through this system. We made experiments for detecting ill-formed patterns. As a result, we show that the system is effective for educational use.

key words　programming style, program pattern, program recognition, diagnosis

1　はじめに

プログラムは、単に仕様通り動けばよいというものではなく、読み易く、わかり易いものでなければならない。プログラムの可読性、理解容易性に影響を及ぼすようなプログラムの書き方を**プログラミングスタイル**と呼び、一般的なガイドラインがいくつか存在する。プログラミングスタイルに従って作成されたプログラムは、プログラムを作った本人ばかりでなく、第３者にとっても理解しやすく、テストやデバックにおける労力を軽減させることができ、保守性の高いものとなる。

プログラム中に構文エラーがあればコンパイラが実行可能コードを作ってくれないし、意味的なエラーがあれば、自分の意図した通りには動作してくれないので、プログラマはデバックを余儀なくされる。しかし、プログラミングスタイルに違反していても、意図した通りにプログラムが動いてくれさえすれば、すぐに不都合が生じるわけではないので、それらの違反はそのままにされることが多い。特に、初心者プログラマは、プログラムのテストを十分にせず、偶然意図した通りに動いただけの場合でも、それに気付きさえしないことがある。

そこで、プログラミング教育においては、文法やアルゴリズムの習得だけでなく、プログラミングスタイルの習得を促すことが重要となる。しかしながら、このようなスタイルに関する教育には、あまり時間が割かれていないのが現状である。この原因の一つとして、スタイルの教育に関する支援システムが実用化されていないことが挙げられる。

我々は、プログラミングスタイルに違反する箇所の検出を行なうことによりプログラミング学習者による悪いプログラミングスタイルの認識を支援するプログラミングスタイル診断システムの開発を目指している。

ここでは、プログラミングスタイルの推奨事例に反しているパターンや、好ましくないプログラムの書き方のことを**悪形パターン**と呼ぶ。我々は、種々の悪形パターンを検出するために、独自にプログラムパターンの表現方法を考案し、これを検出するためのシステムを試作した [1, 2]。我々が提案するプログラムパターンの表現形式では、構造的なスタイルに関するパターンや分散したパターンも記述できる。さらに、個々のガイドライン（各悪形パターン）を一つのプログラムパターンとして表現し、それをデータとして検査を行なうので、検査項目の拡張が容易に行なえ、拡張性に優れている。

本システムは、C 言語のプログラムソースから悪形パターンを使っている箇所の検出を行ない、診断メッセージを表示する。プログラミングエキスパートは、自らの失敗経験等によりいくつかの典型的な悪形パターンの知識を持っていると思われるが、初心者では、それが悪形だと知らずに利用していることも多い。これらの指摘は、プログラムの質の向上だけでなく、プログラマの教育、つまり、プログラミングスタイルにそった良形度の高いプログラムの作成を習得させることにも効果が期待できる。

本論文では、まずプログラミングスタイルについて説明する。次に、検索のための表現形式（プログラムパターン）と、システムの概要について述べる。そして、スタイルガイドラインの記述・認識実験、および、初心者プログラマを対象としたスタイル検査実験に基づく評価について報告する。

2　プログラミングスタイル

プログラミングスタイルには、インデントや変数名の付け方、コメントの方法など体裁に関するものから、構文に関するものなど様々なガイドラインが存在する。

例えば構文に関するガイドラインとしては以下の様なものがある：

[条件式の副作用に関するガイド ライン]: 条件式に代入文を含めるなど副作用のある式は書かない方が良い（特に、論理和や論理積で連結した条件式の場合は注意が必要）。

体裁に関するガイドラインの内、インデントに関しては整形ツールが実用化されており、これの利用により統一されたインデントを学ぶのは容易である。また、変数名の付け方や、コメントに関しては、プログラマ自身が心がけることでわかりやすいプログラムとなる。しかしながら、構文の使い方に関するガイドラインは、初心者プログラマが自然言語で書かれた記述を理解し、自分で検査するのは容易なことではない。さらに計算機システムを利用する場合でも、分散しているパターンや、使われている文脈によって問題となる場合等が

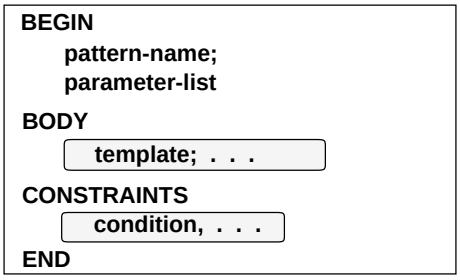

図 1: **プログラムパターンの記述形式**

あり、UNIX の grep に代表されるような正規表現を用いた既存のソースコード探索ツールでスタイルに関する種々の悪形パターンを検出するのは困難である。そこで、本研究では構文に関するガイドラインに焦点をあてた。

プログラム中からこのような悪形パターンの検出を自動的に行なうことが出来れば、初心者でもプログラミングスタイルの検査が容易に行なえるようになり、プログラミング学習、また、プログラムの品質向上に有効に働く。

我々は、初心者プログラマが作成した「ファイルに記述されたデータ 2 組をマージするプログラム（100 行程度）」30 個について、マニュアルでスタイルの検査を行なった。その結果構文に関する悪形パターンが 7 種類見付かり、一番多いものでは 9 割の人が違反していた。また、8 割のプログラムが例で挙げた [条件式の副作用に関するガイドライン] に反していた。このことから、実際のプログラム中には多くの悪形パターンが存在しており、プログラミングスタイルの学習を支援するツールが必要であることがわかった。

3　プログラムパターンの記述形式

3.1　プログラムパターン

本診断システムでは、個々のプログラミングスタイルの悪形パターンを、図 1 に示すような、プログラムパターンと呼ぶ表現形式で記述し、それを利用してプログラム探索を行なう。

2 章で述べた様に周知のプログラムパターンは以下の様な記述上の特徴を持つと言える：1) 命令の単独の使用ではなく、特定の文脈の中での使用を問題としている、2)2 つ以上の命令のプログラム中での離れた場所 (場所については制御フローが存在する以外の条件はない) での利用に関わる場合がある、3) ある命令の存在ではなく、存在しない事を問題としている場合がある。以上の様な特徴を持つパターンを記述し、認識するために以下で述べる方式を考案した。プログラムパターンでは、テンプレートを記述する本体部とそのテンプレートに対する制約条件を記述する制約条件部に分けて表現する。

この様な分離により、1) 文脈に依存したパターンであっても文脈条件は制約条件として述語的に記述できるため、パターン自体の記述は文脈に依存しない構文パターンとして記述でき、またそのような物として認識を行えばよい事になる。制約条件は事後条件として個々に実現すればよく、アルゴリズムもシンプルになり、記述自体の再利用性も増す事になる、2) プログラムのフロー情報などを制約条件として利用する場合でも現在のシステムのアルゴリズムをほとんど変更する事無く、フロー解析ツールおよびその情報を利用する関数の導入だけで済む、という特徴を持つ

また、テンプレートの記述形態からプログラムパターンをベースパターンと複合パターンに分類した。これにより離れた場所に分散して存在するパターンも、ベースパターンの組合せとして表現する事が可能となる。ベースパターンは、検索パターンの基本となるもので、テンプレート部は、C 言語の文法に基づくコードの断片により記述する。複合パターンでは、ベースパターンをテンプレートとして記述する。ベースパターンの再利用により、複雑なパターンの記述を簡単化するねらいがある。また、構成要素であるベースパターンの間に位置の制約を持たないことから、コードが分散しているパターンにも対応できる。

3.2　ベースパターン

ベースパターンの本体部は、C 言語の文法を拡張した記法を使って定義する。構文要素の代わりに、以下のような抽象的なテンプレートとして働くいくつかの正規表現が利用できる。

パターン変数

構文要素のワイルドカード (@:文, :#:式, $:識別子)

構造に関するワイルド カード

カテゴリー名 (@if, @while, @do, @switch, @assign)

クラス名 (@alternate, @loop)

ベースパターンの制約条件部では、テンプレートで利用するパターン変数に対する字句や構文の制約条件やフロー情報に関する制約を定義する。表 1 に制約条件の例を示す。制約条件のパラメータ中のキーワード p,c は、それぞれパターン変数 (p)、定数 (c) をパラメータとして指定することを表す。また説明中の p 等ははパターン変数 p にバインドされたプログラム断片を表す。

type(p,c)	p のデータ型は c である
type_equal(p1,p2)	p1 と p2 の型は等しい
equality_exp(p)	p は等式である
assign_exp(p)	p は代入式である
not_break(p)	p の中には break 文は含まれていない
literal_equal(p,c)	p と c は定数として等しい
constant(p)	p は定数である
integer_large(p,c)	p は整数であり、c より大きい
reach_def(p1,p2)	p1 での定義は p2 での使用に到達する
cflow(p1,p2)	p1 から p2 への制御フローが存在する

表 1: ベースパターンの制約関数の例

3.3 複合パターン

複合パターンの本体部は、以下のように記述されたベースパターンを構成要素とする。

```
label: pattern_name(parameter);
```

複合パターンにおける制約条件は、要素であるベースパターン相互の関係やパターンの位置に関して限定したい場合に利用し、ベースパターンで利用できる制約の他、テンプレート間の位置制約に関するものが含まれる。

複合パターンに固有な制約条件の例を図 2 に示す。パラメータ中のキーワード l はラベル l で指示されるベースパターンにマッチするプログラム断片を意味する。

3.4 プログラムパターンの記述例

条件式の副作用に関するガイドラインのプログラムパターンの記述例、および、そこで利用されているベースパターンを以下に示す。

[条件式の副作用に関するガイドライン]:

node_literal_equal(p1, p2)	p1 と p1 は同じ定数である
node_equal(p1, p2)	p1 と p2 は同じパターンである
loop_body(l)	l は繰り返し文の本体である
condition_part(l)	l は条件文の条件部である
with_in(l1, l2)	l1 は l2 に含まれている
before(l1, l2)	l1 は l2 よりもプログラム中で前にある
not_found(l)	l は存在しない 悪形パターンの特徴 3) より導入
same_variable(p1, p2)	p1 と p2 は同じ変数である

表 2: 複合パターンの制約関数の例

```
BEGIN
side_effect1_pattern;
BODY COMPLEX
  l1:assignment_pattern()
CONSTRAINTS
  condition_part(l1)
END
```

```
BEGIN
assignment_pattern;
BODY
   @assign
END
```

この side_effect1_pattern は、プログラム中の条件式の中で利用されている代入式とマッチする。condition_part(l1) はパターン照合で見つかったプログラム断片 (この例の場合は代入文) が条件式の中に存在する事を主張している。認識したいパターンの記述においては組み込み済みの制約関数とデータベース化してあるベースプランが利用できる。制約関数が不十分な場合は新たな関数を追加することになるが基本的には対象となるプログラム断片 (の解析木) の走査であるので関数自体の実現は容易であり、またシステムへの追加も関数のみの追加であるので簡単にできる。

4 システム概要

本システムは、対象プログラムと診断したいスタイルガイドラインをユーザーに指定してもらい、対象プログラム中からガイドラインに違反する箇所を検出する。システムは UNIX の X ウィンド上で実現しているので、どのユーザサイドからも利用可能であり、また異なった UNIX サイトへの移植も容易である。

本システムの実行画面イメージを図 2 に示す。各種ボタンのあるメニューと、C のソースプログラム、検索プログラムパターン、診断メッセージをそれぞれ表示する 3 つのウィンドウから構成されている。

図 3 に、システムによる診断結果の一例を示す。

システムの構成図を図 4 に示す。本システムは、入力されたプログラムおよびプログラムパターンを構文

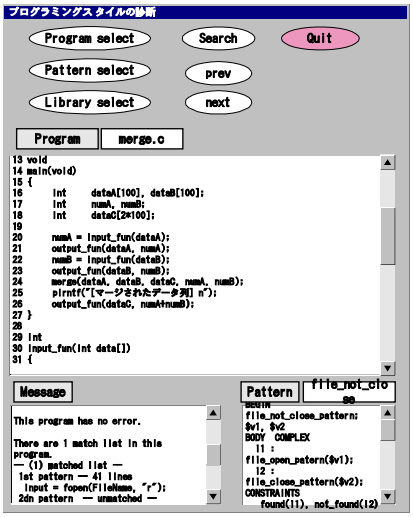

図 2: ユーザインターフェイス

検索 [1]: ファイル処理に関するガイドライン

マッチング箇所が (1) つ見つかりました。

オープンされたままのファイルがあります。ファイル処理において、「ファイルは読み書きを行なう前にオープンし、使い終ったらクローズする」というの基本です。プログラムが終了する時にオープンされたままのファイルは、自動的にクローズされますが、明示的にクローズするように心がけましょう。

——(1) 番目の照合リスト——-
(1) th pattern ——- 41 行目
input = fopen (FileName , ”r”) ;
(2) th pattern == unmatched ==

図 3: 診断結果の例

解析するパーサと、２つのサブモジュール（ベースパターン探索モジュール、複合パターン探索モジュール）から構成される。

ユーザは、まず [Program Select] ボタンを押し、診断対象となるプログラムを指定する。システムは、指定されたプログラムを読み込むと、構文解析を行ない、プログラム解析木を作る。次に、ユーザーは探索したいプログラムパターンを選択する。[Pattern Select] ボタンを押すとポップアップウインドウが現れ、あらかじめパターンデータベースに登録されている検索パターンの一覧が表示されるので、その中から検索パターンを指定する。検索パターンを表示するウインドウはテキストエディタとなっており、ウィンドウ上で直接パターンを編集することも可能である。そして [search] ボタンが押されると、システムが診断を開始する。

プログラムパターンはパーサにより解析され、ベースパターンの場合はベースパターン探索モジュールへ、複合パターンの場合は複合パターン探索モジュールへと、それぞれパターンの情報が渡され、そのパターンにマッチするプログラム断片の情報がそれぞれのモジュールより出力され、それを基に診断メッセージが表示される。また、検索パターンにマッチしたプログラム断片はハイライトされ、[prev][next] ボタンで前後の検出箇所に移ることができる。

ベースパターンの探索では、本体部のテンプレートを木オートマトン化し、プログラムの解析木を対象として検索 (部分木照合) を行い、照合するパターンの全数探索を行なう。このテンプレートの探索部分は、Michigan 大学の S.Paul らが提案する有限状態機械に基づくソースコード探索の枠組 [3] を応用している。そして、受理されたプログラム断片に対して、制約条件部の各条件について検査を行ない、全ての制約条件を満足したプログラム断片 (複数存在し得る) を入力されたベースパターンの検索結果とする。複合パターンの探索では、テンプレートの構成要素であるベースパターンの探索に、ベースパターン探索モジュールを利用する。そして、個々のベースパターンの検索結果の組合せを作り、全ての組合せに対し制約条件の検査を行なう。例えばベースパターンが３つあり、各々に対して５つのプログラム断片が見付かったとすると、その全ての組み合わせ (125 通り) に対して複合パターンの制約条件を調べる。なお更なる詳細に付いて [1] を参照されたい。

診断では、複数の検索パターンをライブラリにあらかじめ登録しておくことにより、複数パターンの一括探索を行なうことができる（[Library Select] ボタンで

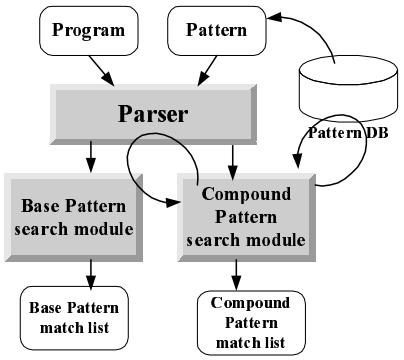

図 4: システム構成図

ライブラリの指定)。これは、あらかじめ決められた複数のプログラミングスタイルに関しての診断を行なう際に有効となる。

5 プログラムパターンの記述および認識実験

プログラミングスタイルの検査においてプログラムパターンの認識に基づく手法が有効であるか評価するために、以下の二つの実験を行なった。一つは、プログラムパターンの記述能力を評価するための記述実験、そして、もう一つは定義したプログラムパターンが、実プログラムにおいて意図した通りのプログラム断片を検索することができるか調査するための認識実験である。

本研究では、現在 C 言語を対象としているが、C 言語は柔軟な構造をもつため、構文上のわずかな違いによりプログラマの意図とは全く異なる動作を行なうプログラムとなってしまうことがある。このような C 言語特有の問題は、悪形パターンの利用からくるものもあり、これらを改め、プログラムを明快に記述することで避けられるものも多い。このような構文上の些細な相違に由来するトラブルの事例を集めた書に、文献 [4] がある。これには、スタイルガイドラインの違反とは少し異なる意味的なバグも含まれるが、可読性、了解性の観点からは密に関係のある問題であり、本実験ではこの文献中の事例を扱う。また、本学のプログラミング演習で利用しているプログラミングスタイルの項目も対象とする。文献 [4] では、トラブルの事例を、7 つのカテゴリ（語彙的、構文的、意味的、リンケージ、ライブラリ関数、プリプロセッサ、移植性）に分類して集めてある。本実験では、このうち構文的、意味的な落し穴を対象としプログラムパターンの記述および認識を行なった。

また、認識実験では、プログラミングスタイルとして、記述実験で定義したプログラムパターンを利用する。診断対象プログラムは、UNIX の GNU 版テキストユーティリティのパッケージプログラム群 (cat, head, wc 等) の 22 種類のプログラムとする。

5.1 記述実験

文献 [4] 中の構文的な落し穴では、全 6 種類中 5 種類の項目を定義することができた。データ化を行なったのは、演算子の優先度、セミコロンの問題、break 文のない case ラベル、ぶらさがり else 問題の 5 種類についてで、12 個のプログラムパターンを定義した。データ化が行なえなかったのは、「複雑な宣言は、typedef を使って簡単にする方がよい」という宣言の書き方に関するもので、現在システムにおいて宣言部分を対象としたパターンの認識を実現していないため扱えない。ただし、パーサによる解析は行なっているので拡張可能である。

意味的な落し穴については、悪形事例を示している 7 種類の内、3 種類（ポインタ変数の利用、数え上げと非対象な境界、評価順序）の項目について 7 個のプログラムパターンを定義した。定義を行なっていない内の 3 種類（代入における型の不一致、ポインタ同士の代入、整数のオーバーフロー）については、構文的な落し穴で扱えなかったのと同様の理由である。また、残りの 1 種類（main からの返却値の有無）については、関数単位にその返却値の型と値を調べ、その整合性を検査するというもので、プログラムパターンとして定義するのには不向きであると考えた。但し本システムでの記述は可能である。

この他、本大学のプログラミング演習において利用しているプログラミングスタイルのうち、構文レベルの悪形パターンについて、19 個のプログラムパターンを定義した。

5.2 認識実験

テキストユーティリティの 22 種類のプログラムに対し、記述実験で定義した 38 個のプログラムパターンの検索を行なった結果、1 つのプログラムにおける延べマッチング数は平均 45.7(最大 162、最小 7)、また、1 つのプログラムパターンの延べマッチング数は平均 26.5(最大 195、最小 0) であった。なお、対象プログラムのコメントを除く行数は平均 497 行（最小 107 行、最大 1354 行）である。

5.3 記述、認識実験に基づく考察

本実験より、テキスト上の単純なパターンマッチングではマッチングが難しい各種パターン（文脈依存なパターン、複数の関数にまたがるパターン、欠如したものを含んだパターン等）をプログラムパターンとして容易に記述が行なえ、探索可能であることが確認できた。これより、本システムがプログラミングスタイルの悪形パターンの検出に有効な手法であると言える。

プログラムパターン定義者の意図したプログラム断片以外のものとのマッチング（ミスマッチ）の問題は、制約の調整で対処できた。また、ガイドラインによっては新たな制約が必要となる場合があったが、制約条件の追加がライブラリ関数の追加として容易に行なえた。このことからシステムの拡張容易性が確認できた。

6 他のスタイル検査システムとの比較

プログラミングスタイルの検査ツールとしては、プログラミングの初期教育におけるサポートを目的とした Pascal プログラムの自己評価ツール CAP [5] や、プログラムのモジュール性、構造、レイアウトなど 6 つの視点から品質の検査を行なう STYLE [6] などが開発されている。これらの研究では、種々の視点からプログラムのスタイルに関する考察をおこなっているが、実現されたシステムでは、構造的なスタイルや文脈に依存した要素に関する検査は対象としておらず、インデントやコメントに対する検査や、モジュールの大きさなどの統計量に関する検査が中心となっている。また、各々の検査が手続き的に組み込まれているので拡張性に問題がある。これに対し、本手法では検査項目をプログラムパターンとして追加するだけで新たな項目に対する検査が可能なので、拡張性に優れている。

また、構文パターンマッチングに基づき C プログラムの落し穴検出を行なう Fall-in C[7] では、パターンを木として表現しているために、人間がパターンを見た時にそれを直観的に理解しにくい。このため、複雑なパターンの記述をしようと思うと、表現が複雑になってしまうが、我々のプログラムパターン表現では、パターンをテキストイメージそのままで定義できるので、新たなプランの表現が容易であり、可読性にも優れている。さらに Fall-in C のように解析木をマッチングのベースとした場合、複数の関数に渡るパターンや、欠如したパターンには対応できないが、本手法では複合プランを用いることにより認識可能である。

プログラムの構造をシステムがガイドするツールに構文指向または構造指向エディタと呼ばれるものがある。しかしこれらは構文的まとまり (文法論的には文脈自由文法での構造) の簡易入力または学習を目的とするもので、本システムで対象としているような文脈依存文法に基づく構造を対象としてものではない。またより上流の構造を考えるものにデザインパターンがあるが、これは個々の詳細な実行文については考慮しない。

7 初心者プログラマによる評価実験

初心者プログラマの場合、プログラミングスタイルおよび文法等に関する知識が不足していることから、悪形パターンを認識できないことが多い。本システムは、悪形パターンの検査支援を行なうだけでなく、プログラマのプログラミングスタイルの検査能力を向上させることによって、ツールに頼らなくても悪形パターンを認識できるるようにすること、つまり、プログラミングスタイルを習得させることも目的としている。既存の検査ツール [5, 6, 7] では、検査能力の評価を行なっているものはあるが、このような教育効果についての評価を行なっているものはない。我々は、本プログラミングスタイルの診断システムの利用を通して、初心者プログラマのスタイル検査能力を向上させることができるか調査することを目的とした実験を行なった。

7.1 実験概要

システムの有効性を評価するために被験者を 2 つの組に分け、片方の組のみシステムを利用させる。そして、その組におけるシステムを利用する前と後の検査

結果の比較、および、初回と数回システムを利用した後でのシステムを利用しない場合の検査結果の推移を、まったくシステムを利用しなかった被験者の組の推移と比較する。

まず、被験者を 2 組のグループ（A 組, B 組）に均一に分けるために、簡単なプログラミングに関する能力テストを行なう。そして、テスト結果に基づき、なるべく同等のレベルになるよう組分けをする。そして、いくつかの指定されたプログラミングスタイルの違反をソースプログラム中から検出し、修正してもらうという実験を問題 1、問題 2、問題 3 の順番に、以下の要領で行なう。実験では、プログラムとプログラミングスタイルの記述された用紙を配布し、検出、修正は直接プログラム上に書き込むものとした。

1. [問題 1]: 被験者全員（A 組, B 組）が、システムを利用しないで検査を行なう。その後、A 組のみ再度プログラミングスタイルの診断結果を参照しながら検査を行なう。

2. [問題 2]: A 組は、プログラミングスタイルの診断結果を参照しながら検査を行ない、B 組はシステムを利用しないで検査を行なう。

3. [問題 3]: 問題 1 と同様。

被験者は、本学情報工学科の 4 年生 9 人である。

検査対象とした**プログラム**は、問題 1 が二つのファイルに記入されたデータのマージ、問題 2 が KWIC(Keyword in Context) による索引付け、問題 3 がクイックソートで、それぞれ 100 行から 130 行程度のプログラムである。これらのプログラムに、プログラミングスタイルの違反を 5 種類埋め込んだものを検査対象のプログラムとした。

検査するよう指示した**ガイドライン**は、プログラム中に埋め込んだ 5 種類の違反に関する以下のガイドラインと、for 文の利用に関するガイドラインの合計 6 つである。

1. 定数に関するガイドライン
2. 条件式の副作用に関するガイドライン
3. インクリメント/デクリメント演算子の利用に関するガイドライン
4. ファイル処理に関するガイドライン
5. 条件節に関するガイドライン

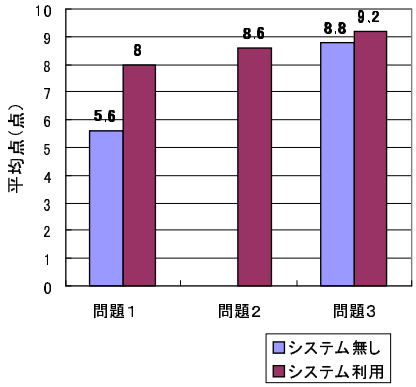

図 5: A組の平均点

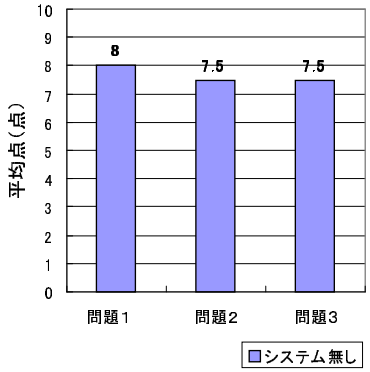

図 6: B組の平均点

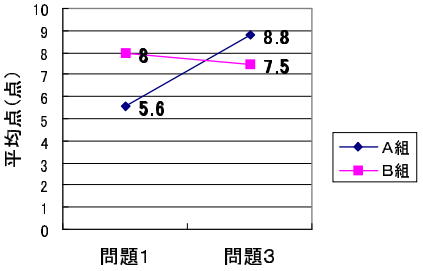

図 7: システムを利用しない検査での平均点の推移

7.2 実験結果および考察

まず、プログラミングに関する能力テストの結果により、なるべく 2 つの組が同等のレベルになるよう被験者 9 名を A 組（5 人）と B 組（4 人）に組分けした。プログラムの穴埋め問題（10 点満点）で、平均点が A 組 9.0 点、B 組 8.2 点であり、ほぼ同等のレベルであるとみなせる。

次に、それぞれの問題について、悪形パターンの箇所を正しく検出し、修正しているかについて 10 点満点で採点を行なった。A 組、B 組の平均点をそれぞれ図 5、図 6 に示す。また、システムを利用していない状況での A 組、B 組の問題 1 と問題 3 の平均点の推移を図 7 に示す。

図 5 を見ると、A 組の被験者はシステムを利用することにより平均点が上がっていることがわかる（特に、問題 1）。また、図 5 と図 6 を比較してみると、問題 2 以降において A 組の方が B 組より平均点が高くなっている。これより、本システムが初心者プログラマの悪形パターンの検出の支援に有効であると言える。

そして図 7 を見ると、B 組の平均点は問題 1 と問題 3 でほとんど変化が見られないが (-0.5 点)、問題 1、問題 2 においてシステムを利用した A 組では、問題 3 の平均点の方が問題 1 の平均点よりだいぶ高くなっている (+3.2 点)。これより、A 組の被験者は、システムを利用したことにより、悪形パターンの検出における能力が向上したと考えることができる。

8 おわりに

C 言語のソースプログラム中からプログラミングスタイルに違反する箇所を検出するプログラミングスタイルの診断システムを構築した。本システムは、検索したいプログラム断片をプログラムパターンとしてデータ化し、これを利用してソースコードの探索を行なう。

本システムが、プログラミングスタイルの種々の悪形パターンの検出に有効であることを確認した。

プログラミング演習において、これまでは教師が学生の作成したプログラムを個々にチェックし、面談等で個別に注意するしか方法がなかったが、本システムを利用することにより、教師が介在しなくても、ある程度自分でプログラミングスタイルの検査が可能となる。実験を通して、本システムが初心者プログラマのプログラミングスタイルに関する悪形パターンの検出、および修正の支援に有効に働くことを示した。さらに、本システムの利用が、初心者プログラマのプログラミングスタイルの悪形パターンの検査能力の向上に役立つことが確認できた。また、プログラミングスタイルの検査を習慣づけることで、プログラミングスタイルに関する意識が向上し、良形度の高いプログラムの作成が身に付くと期待できる。

これまでは、文献等に載っているプログラミングスタイルを中心にプログラムパターンとしてデータ化し、システムのテストを行なってきた。今後は、教育効果をさらに高めるために、実際に初心者プログラムの作成したプログラムを対象に、どのような悪形パターンが多く使われているかを分析し、初心者プログラマの教育において有効となるスタイルガイドラインのデータ（プログラムパターンデータベース）を充実させていくことが必要である。

参考文献

[1] Rika Sekimoto and Kenji Kaijiri: *"Plan representation and its recognition approach for program recognition"*, JCKBSE'96, pp.198–201 (1996)

[2] Rika Sekimoto and Kenji Kaijiri: *"A detection of ill-formed patterns about programming style"*, JCKBSE'98, IOS Press, pp. 165–168 (1998)

[3] S. Paul and A. Prakash : *"A Framework for source code search using programming patterns"*, IEEE Trans. Software Eng., Vol.20, No.6 (1994)

[4] Koenig A. (中村 明 訳): *"C* **プログラミングの落し穴***"*, (株) トッパン (1990)

[5] Tom Schorsch: *"CAP: An automated self-assessment tool to check Pascal programs for syntax, logic and style errors"*, SIGCSE'95 (1995)

[6] Al Lake and Curtis Cook: *"STYLE -An automated program style analyzer for Pascal-"*, SIGCSE Bulletin, Vol. 22, No. 3 (1990)

[7] 小田 まり子, 掛下 哲郎: *"***パターンマッチングに基づいた** *C* **プログラムの落し穴検出方法***"*, 情報処理学会論文誌, Vol.35, No. 11 (1994)