短冊型問題を用いた プログラミング学習アドバイスツール

久野　靖[1,a)]

概要：筆者らは 2017 年度から、プログラミング入門科目の試験において、完全自動採点の可能な「短冊型問題」(プログラムを行単位でばらばらにして記号を付した選択肢とし、正しいプログラムに対応する記号列を解答させる問題形式) を用いており、その練習のため毎授業時冒頭に「確認テスト」を実施している。一方、学生の演習のようすを見ると、end の過不足、if 文の外の else の存在、return の後に必要な文を置く、未定義変数の参照など、「基本的な間違い」が多く見られ、このような基本的な間違いを犯さないための学習支援の必要性を感じていた。そこで、正誤を見る試験とは別に、選択肢を並べてプログラムを完成させた状態で「アドバイス」ボタンを押すと基本的な間違いを指摘してくれるツールを作成し、2019 年度から使用を開始した。本稿では、このツールの設計と実装および現時点での評価について述べ、このようなツールの有用性について検討する。

キーワード：プログラミング学習, 短冊型問題, アドバイス, 基本的な誤り

1. はじめに

短冊型問題 [1] ないし Parsons Problem[2] とは、プログラミングの試験に用いることのできる問題形式であり、正解プログラムを行単位でばらばらにして、(必要なら誤答の行を追加した上で) 並べ替えて選択記号を付し、選択肢とする出題形式である。回答者には、PBT(Paper Based Test、ペーパーテスト) では解答欄に記号列を記入させるが、CBT(Computer Based Test、コンピュータを用いたテスト) では画面上で選択肢の行を並べ替えて解答プログラムを構成させる形も取れる。

この問題形式によるプログラミング試験のスコアは、広く行われている「紙にプログラムを書かせる」「入力欄にプログラムを打ち込ませる」試験と相関を持っている [3][4][5]。実際、ある問題に対して正解プログラムを書く能力が無ければ、同じ問題を用いた短冊型問題に対して正解することも難しい、というのが筆者らの経験である。

そして、解答が記号列であることから、想定正解 (複数あってよい) との一致を見ることで自動採点が可能である。筆者らは、電気通信大学の 1 年次必修科目「基礎プログラミングおよび演習」(履修人数約 800 名、以下「本科目」と記す) において、期末試験をプログラミング作成問題のみの内容で実施するために、短冊型問題の CBT を採用している [6][7]。このような大量の問題の自動採点が可能になることは、短冊型問題の利点である。

採点に際しては、想定正解と解答記号列の編集距離 (1 文字挿入、1 文字削除、隣接文字交換の操作で両者を一致させる場合の最小操作数) を用いている。複数の想定正解に対して編集距離を計算し、

[1] 電気通信大学
[a)] y-kuno@uec.ac.jp

その最小値が 0 であればいずれかの想定正解に一致しているため 2 点、最小値が 1 であれば部分点として 1 点を付与している。これによって、微細な誤りをある程度救済できる。

短冊型問題は一般的な形式ではないため、本科目の初回を除く毎授業時冒頭 10 分間で、短冊型問題を用いた「確認テスト」と呼ばれる練習問題各 2 問を課している (成績には算入しない)。確認テストの問題は期末試験問題の類題であり、学生にもそのことを告知している。

本科目の演習で学生が書くプログラムを見ていると、次のような「基本的な誤り」が含まれていることが多い (使用言語は前半が Ruby、後半が C であるが、ここでは Ruby の用語で述べる)。

- end の個数が多すぎる/少なすぎる。
- if 文の外なのに else や elsif が書かれている。
- return の後 (実行されない位置) に必要と思われる計算が書かれている。
- 未定義 (未代入) の変数が参照されている。

これらの基本的な誤りは構文エラー、実行時エラー、必要な計算の未実行につながるため、これらが含まれた状態でプログラムを完成させることはできない。初心者はそのことが分からず、練習を通じて (また教員・TA の指導によって) これらの誤りを犯さないように学んで行くが、できるだけ速やかにこれらの誤りを犯さないようにできれば、本人や教員・TA の負担を軽減できるはずである。

そして、短冊型問題の枠組を活用することで、上記のような基本的な誤りを検出し、分かりやすい表現で学習者に伝えることが可能なのではと考えた。このアイデアに基づき開発したツールを、筆者らは「短冊アドバイザ」と呼んでいる。

短冊アドバイザは 2019 年夏に開発し、2019 年度後期 (10 月〜) の授業において試行を行っている (図 1 に出力のようすを示す)。利用方法としては、CMS の各回の教材中に「自由に利用できる教材」として含め、その利用は各学生に任せている。

以下本稿では、第 2 章で短冊問題のこれまでの活用状況について述べた後、第 3 章で短冊アドバイザの設計と実装を説明する。続いて第 4 章で、

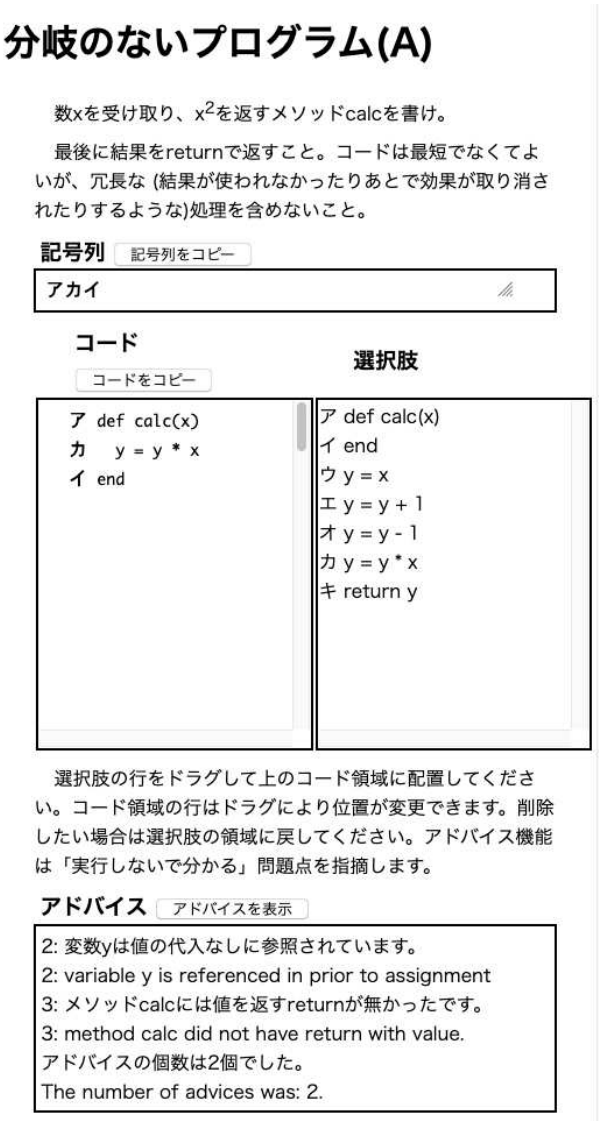


図 1　アドバイスを出力したようす

現時点までの使用に基づく知見を述べ、基本的な誤りの軽減に対する効果について検討する。最後に第 5 章でまとめと将来展望について述べる。

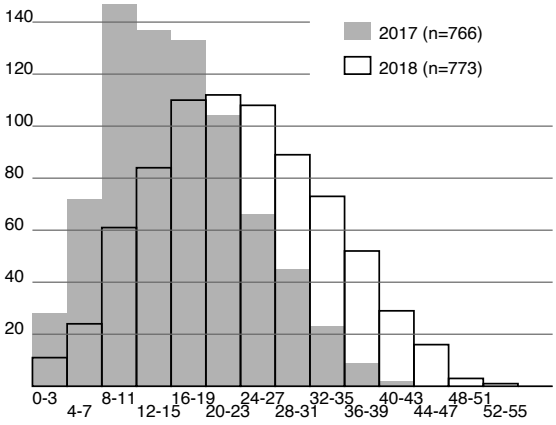
2. 短冊問題の活用経験

2.1 短冊問題 CBT の開発と採用

短冊問題は筆者らが 2012 年に、採点の容易なプログラミング能力試験を作る意図で、情報入試研究会における作問活動の一環として開発した [1]。[*1]

[*1] 当初は Parsons らの研究 [2] は認識していなかった。

図 2　基礎プログラミングおよび演習の試験素点分布



情報入試研究会のテストは PBT であり、短冊問題も紙に選択肢や解答欄を印刷して実施していた。PBT で回答する場合は、下書き用紙などに選択肢を見ながら回答プログラムを作成し、完成したあと対応する選択記号を回答欄に記入するという過程が必要になる。

CBT であればそのような 2 度手間は不要で、直接画面上で選択肢をドラグして並べ、回答プログラムを構成できると考え、そのようなツールを試作した。このツールはブラウザ上で JavaScript を用いて動作するものであった。

その後、情報入試研究会の活動を基に大阪大学 (代表機関)・東京大学・情報処理学会 (連携機関) が協力して大学入学者選抜改革推進委託事業 [8] を実施する際、そこで開発する CBT の中に短冊問題の機能が盛り込まれ、試作試験に複数の短冊問題が含められた。

筆者は 2016 年度から電気通信大学に移り、初年次情報教育の責任者となったが、そこでは 1 年次必修科目「基礎プログラミングおよび演習」について、多くの学生がきちんとプログラムを書けるようになっていないという問題があった。

これに対処するため 2017 年度から科目設計を見直したが [6][7]、その一環として「プログラムが書ける」ことの重視を明確に示すため、期末試験問題をすべてプログラム作成の問題とた。その際、800 人ぶんの答案を短期間で採点する必要があることから、短冊問題を採用した。

もともとこの科目の授業は LMS(学修支援システム) として Moodle を採用しており、期末試験も CBT 化されていた。短冊問題の機能はもちろん無いため、上記の試作ツールを採用し、受験者はツール画面で問題を解き、表示されている選択肢の列を Moodle 試験モジュールの解答欄にコピーペーストして提出させ、収集した解答を Moodle から取り出し別プログラムで採点する形で運用した。

2017 年度はこのような形で実施したが、受験者にとっても採点者にとっても処理が繁雑なため、Moodle の拡張機能として短冊問題に対応した試験の出題 (画面上の並べ替え機能を含む) と採点を行うモジュールを改めて開発し、2018 年度以降はこちらを使用している。いずれの年度も試験は 80 分間で 28 問を提示し、「実際にプログラムを書ける学生がよい成績を取る」自動採点 CBT が実現できているものと考えている (図 2)。[*2]

2.2　短冊問題の記述形式

短冊問題を画面で並べ替えて解答するツールを最初に作ったときに、問題を記述する形式が必要となった。ツールが生成するのは問題を表示する HTML ページに選択肢と解答欄 (そこに選択肢を個別にドラグして並べる) が埋め込まれたものであり、必要なのは、HTML による問題記述と、選択肢の情報だけである。そこで、テキストファイルの前半に HTML で問題を記述し、「`@@@`」で区切ってその後の各行に選択肢の行を置くこととした。図 3 に記述ファイルの例を示す。この形式は後から開発した Moodle モジュール版でも同一である。

プログラムを構成する際に字下げが正しくできないと解答しづらいので、各選択肢の末尾に次の記述を加えることで字下げを制御する。

- `// 1+` — 次の行から字下げを 1 レベル増やす。
- `// -1` — この行から字下げを 1 レベル戻す。
- `// -1 1+` — この行から字下げを 1 レベル戻すが、次の行から再び 1 レベル増やす。

C 言語でいえば 1 番目は関数の先頭行、if、while

[*2] 2018 年度の方が点数が向上しているのは、カリキュラムの改良と難しすぎる問題を手直ししたことの 2 点が理由であると考えている。

```
<h1>分岐のないプログラム (A)</h1>
<p>数 x を受け取り、x<sup>2</sup>を返すメソッド
calc を書け。</p>
<p>最後に結果を return で返すこと。コードは最短で
なくてよいが、冗長な (結果が使われなかったりあとで
効果が取り消されたりするような) 処理を含めない
こと。</p>
@@@
ア def calc(x) // 1+
イ end // -1
ウ y = x
エ y = y + 1
オ y = y - 1
カ y = y * x
キ return y
```

図 3　短冊問題の記述ファイル

などの制御構文の行に指定し、2 番目はこれらの範囲の終りを表す「}」のみの行に指定し、3 番目は else-if の行に指定することを想定している。

現在使用中の言語は C、Ruby、JavaScript、疑似コードなどがあり、どれも上記の 3 つで字下げを記述できている (この指示で対応できるようなコーディングスタイルで選択肢を用意。たとえば C や JavaScript であれば, if や while の中の文は必ず{…}で囲むスタイルを採用している)。[*3]

3. 短冊アドバイザの開発

3.1 開発の方針

前述したように、本科目の演習で学生が書くプログラムに end が足りないなどの「基本的な誤り」が多く見られることから、これらが誤りであることを早期に認識し、避けられるようになるための学習ツールとして、短冊型問題ツールを拡張した短冊アドバイザを構想した。短冊アドバイザの開発に当たっては、次のことを方針とした。

(1)　単独の Web ページとして動作する。
(2)　「基本的な誤り」の指摘に専念。
(3)　個々の問題の採点はおこなわない。
(4)　アドバイスの記録を取れるようにする。

(1) については、ツールとして手軽に使え、設置などの手間を軽減したいことによる。(2) については、多くの機能を盛り込むことによる複雑化を避けてこのようにした。(3) については、単独のページで動作し、かつ採点を行うと、ページソース内に正解を含める必要があり、学生がそれを取り出して活用する (結果として問題を考えてもらえない) ことを避けるためである。(4) については、ツールがどれくらい使われているか、どのようなアドバイスが表示されているかをチェックでき改良に役立てるためである。

次に、アドバイスをどのように生成するかを決める必要があった。1 つの方法は明らかに、組み立てられたソースコードを通常の言語処理系と同様に解析し誤りを検出することである。

しかし、言語処理系の解析部と同等のものを実装し組み込むのは大変であるし、「基本的な誤り」の検出に限るとしたら大げさな仕掛けになりすぎる。さらにこの方法では、選択肢の切れ目などの情報は利用しないことになる。もっと「用意した選択肢に即した」形でアドバイスを用意できないかと検討し、次の方針を採用した。

- 問題記述の選択肢 (回答コードの 1 行ぶん) ごとに、その行は何の動作をしているかを、「基本的な誤り」の指摘に必要な範囲で付加情報として記述する。
- コードを組み立て終った時点で「アドバイス」ボタンを押すと、その時点で並んでいる順に付加情報をスキャンして未定義等の検出をおこなう。
- すべての誤りを検出する必要はないので、何を付加情報として記述し、どの程度まで検出するかは問題ごとに (付加情報の記述により) 違えられるようにする。

たとえば、まったく付加情報を記述しなければ、何の検出も行われず、単に並べ替えでコードを作成して見られるだけの動作となる (通常はそのような使い方はしないが)。あくまでも目的は「基本

[*3] Python のように字下げで制御構造を表す言語では「範囲終り」のようなコメント行を選択肢に加える必要がある。画面上でインデントを操作するインタフェースを付加する方法も考えられるが、実装していない。

```
ア def calc(x) // 1+
  enterproc('calc', true); defvar('x');
イ end // -1
  leave();
ウ y = x
  act(); refvar('x'); defvar('y');
エ y = y + 1
  act(); refvar('y'); defvar('y');
オ y = y - 1
  act(); refvar('y'); defvar('y');
カ y = y * x
  act();refvar('x');refvar('y');defvar('y');
キ return y
  act(); refvar('y'); doreturnval();
```

図 4　アドバイザ用指定を追加した選択肢記述

的な誤り」がよくある形で含まれていたときにそれが検出できアドバイスを表示できるようにすることである。

なお、「変数が代入されているが参照されない」のは一般にはエラーではないが、課題でプログラミングを演習している際には疑わしい記述であるため、検出できるようにしている。このこともあり、検出表示をエラー等ではなく「アドバイス」と呼んでいる。

3.2　付加情報の記述とアドバイザの構成

付加情報の記述例を図 4 に示す。付加情報は宣言的な記法を作成してもよかったが、今回は JavaScript のコード片そのものであり、その中に情報登録のための API 呼び出しを含めることで情報を記述している。アドバイス生成時にこれらの付加情報コードを選択されている順に実行することで検出を行う。

実装では、これらの付加情報は行ごとに 1 つの JavaScript 無名関数に変換してハッシュ表に登録し、アドバイスボタンが押された際に選択記号に基づいて順次これらを取り出し実行し、誤りの検出に応じてアドバイスを出力する。最後の記号まで動作が終ったら、end の不足やグローバル変数の代入がないなど全体として分かるアドバイスを出力する。

多数のアドバイスで使用者が圧倒されないように、1 回の実行 (アドバイスボタン押し下げ) で表示されるアドバイス数は最大 5 個としている (6 個以上あるときは 6 個目として、もっとあるが省略という表示をおこなう)。留学生にも配慮して、すべてのアドバイスは和文と英文のペアで (2 行で) 表示する (図 1)。

記録は、アドバイスボタンが押されるごとに特定の URL にユーザ ID と選択記号列を HTTP GET メソッドで送信することで行う。GET の送信はページ内に目立たなく (極めて小さく) 配置された iframe 要素の src 属性に URL 文字列を代入することで行っており、サーバにアクセスできない場合もツールの動作には影響しない。ユーザ ID は Moodle の設定でアドバイザのページ呼び出し時に URL の末尾に付加することができるので、これを用いている (この付加情報が無い場合は無いという印の ID を送信)。

表 1　Ruby 用の情報記述 API

記述	説明
`act()`	動作を行う文である (メソッドの外にあってはならない)。
`dervar(`v`)`	変数 v に値を代入している (またはパラメタとして値を受け取っている。
`dobreak()`	break 文である。
`donext()`	next 文である。
`doreturn()`	値を返さない return である。
`doreturnval()`	値を返す return である。
`enterclass(`n`)`	クラス n の定義開始。
`enterelse()`	if 文の中の else である。
`enterelsif()`	if 文の中の else-if である。
`enterif()`	if 文の開始である。
`enterloop(`k`)`	種別 k のループである (k はループの種類を表す文字列)。
`enterproc(`n`, `r`)`	メソッド n の定義開始。r は論理値で、メソッドが値を返すか否かを指定する。
`leave()`	制御構造 (メソッドを含む) の終わり。
`refvar(`v`)`	変数 v の値を参照している。

3.3　Ruby 用の記述 API

付加情報記述に使用する API は問題の記述に使用する言語ごとに別のものを使用する。これは言語ごとに用語が異なることから必要である (たとえば C 言語の「手続き」は Ruby では「メソッド」となる)。本科目では前半 (入門部分) に Ruby を採用していることから、まず Ruby 言語用の API (表 1) を開発した。

見て分かる通り、ほとんどの呼び出しは「この行が何の文か」を示すものとなっている。これは、各行のコードを解析しないと決めたためやむを得ない。これに基づいて else、break 等の位置がおかしかったり return が無いなどの誤りを検出する。また、クラス、メソッドの入口ではそのスコープの記号表をスタックに積み、`leave()` それらを取り降ろすことで、スコープに対応した変数のチェックに備えている。

`act()` は実行文であることを示し、return の後に `act()` のついた行があるとおかしいことが分かる。残りは `defvar()`、`refvar()` であり、変数の代入と参照を意味する。これらに応じて記号表の情報を操作することにより、未定義変数の参照や定義変数の未参照を検出できる。Ruby では変数名冒頭の`$`や`@`で種別が分かり、これらの種別に応じた対応を行う。

表 2 に Ruby 用のアドバイス文の一覧を示す (記号は分析時の参照に使用するために付した)。

3.4 C 言語用の記述 API

C 言語では変数や関数に対して宣言が必須であるため、記述命令の形が同じでもチェックの内容には違いがある。たとえば Ruby 版では変数定義は代入によっておこなっていたが、C 言語では宣言により定義され、代入の情報は記述しない。また関数宣言、型 (や struct) 定義、include の記述もおこなう。表 3 に C 言語用の API を示す。

C は強い型の言語であり、型に関するエラーが静的に検出できるが、今回の API では対応しなかった。これは、型に関するエラーはここまでに扱って来た「基本的な誤り」よりは重要でないと考えたことによるが、必要ならたとえば整数型と実数型の使い分けに絞った検出のみを行うことも考えられる (C 言語で扱えるすべての型の静的検査に対応するのはおそらくやりすぎと思われる)。

4. 短冊アドバイザの評価

4.1 評価の枠組み

冒頭で説明したように、短冊アドバイザは 2019 年度「基礎プログラミングおよび演習」において試行として扱い、利用するかどうかを学生に任せている。そこで、利用している学生としていない学生を比較する形で評価を行った。

表 2　Ruby 言語用のアドバイス文

記号	説明
a	メソッド x の定義が他のメソッド y の中にあります。
b	変数 v は値の代入なしに参照されています。
c	メソッドの外に実行する文があります。
d	実行されない文があります。
e	メソッド中でないのに return があります。
f	値を返すメソッドなのに return に式がないです。
g	ループを直ちに終らせてしまう return があります。
h	値を返さないメソッドなのに return に式があります。
i	ループを直ちに終らせてしまう return があります。
j	ループの中でないのに break があります。
k	ループを直ちに終らせてしまう break があります。
l	ループの中でないのに next があります。
m	if の中でないのに elsif があります。
n	if の中でないのに else があります。
o	この if 文では既に else が現れています。
p	余分な end があります。
q	メソッド m には値を返す return が無いです。
r	局所変数/パラメタ v は一度も使われていません。
s	インスタンス変数 v は一度も代入されていません。
t	end の個数が不足しています。
u	グローバル変数 v は一度も代入されていません。

授業は本稿執筆時点で進行中であるので、11/25 時点ですべてのクラスが完了している第 6 回までのデータ (複数回受験可能なので初回送信結果に限定) を用いる。昼間課程の学生 (12 クラス、Moodle 登録人数 801 名) のうち、確認問題 10 問 (第 2〜6 回に各 2 問で実施) すべてに回答し、なおかつ無回答または「選択記号 1 文字だけの解答」がなく、2 問合計の回答時間が 30 秒以上である学生 460 名を分析対象とした。[*4]

短冊アドバイザの使用記録は前述の方法で問題ページを格納した HTTP サーバの request-log か

[*4] 学生のなかには、開始してすぐにでたらめな回答を送信して解説 (1 回送信すると見られるようになる) を見たり、それを覚えて再度受験し満点を取るなどの行動をする者が一定数見られたので、それらを除外するためこのようにした。初回送信で 2 問合計回答時間が 30 秒未満ものもはすべて採点結果が 0 点であった。

表 3　C 言語用の情報記述 API

記述	説明
act()	動作を行う文である (関数の外にあってはならない)。
defproc(n)	関数 n の宣言。
definc(i)	ファイル i を include。
defvar(v)	変数 v を宣言している (パラメタ定義を含む)。
deftype(t)	型指定 t を定義。
dobreak()	break 文である。
docontinue()	coninue 文である。
doreturn()	値を返さない return である。
doreturnval()	値を返す return である。
enterloop(k)	種別 k のループである (k はループの種類を表す文字列)。
enterproc(n, r)	関数 n の定義開始。r は論理値で、メソッドが値を返すか否かを指定する。
enterif()	if 文の開始である。
enterelsif()	if 文の中の else-if である。
enterelse()	if 文の中の else である。
leave()	制御構造 (関数を含む) の終わり。
refinc(i, f)	関数 f(ファイル i の include 必要) を使用。
reffunc(f)	関数 f を使用。
reftype(t)	型定義 t を使用。
refvar(v)	変数 v の値を参照してる。

ら取得した。対象期間は 2019.10.1〜11.24 の 55 日間である。解答時間は問題ページを読み出した時点からアドバイスボタンを押して解答記号列が送信された時点までの経過時間で求めた。2 回以上アドバイスボタンを押している場合は、前のアドバイスボタンからの経過時間を使用し、ただし直前と解答記号列が同一のものはボタンを連打しただけなので無視した。

1 つの問題について、問題を取得し、1 回以上アドバイスボタンを押したものを 1 つのセッションと呼ぶことにする。当該期間のセッション数は 683、アドバイス回数は 1393 回で、平均して 1 セッションあたり 2.0 回のアドバイス回数があることになる。[*5]

1 回以上のセッションが見られた学生数は 176 名であったが、うち 72 名は 1 回だけ、27 名は 2 回だけであった。これらは「たまたま」短冊アドバイザを使って見ただけと考えて除外し、3 回以上

[*5] このほかに問題を見るだけで 1 回もアドバイスボタンを押していないものが 843 件あった。これは問題を練習問題としてのみ使用しているものと想像される。

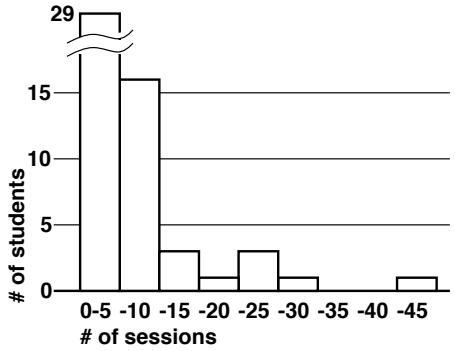


図 5　3 セッション以上の分析対象学生のセッション数分布

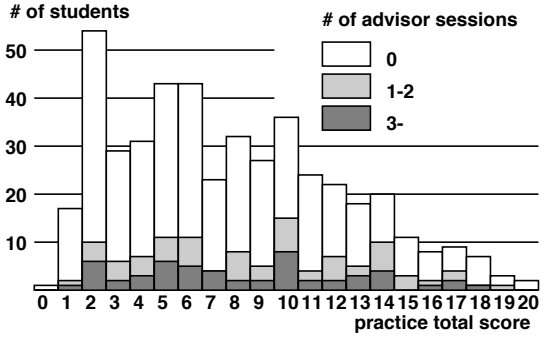


図 6　確認問題の得点分布

の 77 名を考える。さらに、176 名のうち 60 名 (そして 77 名のうち 23 名) は確認問題の未解答があるため分析から除外する。その結果、3 セッション以上ある分析対象学生は 54 名となった。図 5 に 54 名についてのセッション数の分布を示す。最も多かった学生のセッション数は 43 であったが、大半は 10 以下であったことがわかる。

図 6 に確認問題合計点の分布を示す。各問題は正解 2 点、部分点 1 点であるので満点は 40 点であるが、最高点はその半分の 20 点であった。全体の分布は 2 点、5-6 点、10 点付近にピークのある山形となっている。また、アドバイザを使わなかった (セッション数 0 の) 学生とセッション数 1〜2、3 以上の学生を区分して描いてあるが、これらの間で目立った違いはなさそうに見える。

4.2　アドバイス相当数の分析

短冊アドバイザの期待される効果は、end の過不足などの「基本的な誤り」について速やかに学習し、このような誤りを犯さなくなることである。

表 4　アドバイス相当数の比較

回	利用 (n=54)	非利用 (n=344)	検定結果
#2	0.48 (0.75)	0.56 (1.03)	p = 0.53
#3	6.37 (1.26)	7.20 (2.96)	p = 0.00054**
#4	1.11 (1.40)	1.74 (1.81)	p = 0.0040**
#5	1.59 (1.98)	2.07 (1.94)	p = 0.10
#6	2.07 (2.21)	2.57 (2.51)	p = 0.14
合計	11.6 (4.85)	14.1 (6.44)	p = 0.0012**

数字は 1 人当たり平均 (かっこ内は標準偏差)

そこで、アドバイザ利用者 (セッション数 3 以上、以下同様) と非利用者の確認問題解答について、このような誤りの個数を比較する。

誤りの個数を求める方法としては、短冊アドバイザの検出部分を (本稿のデータ処理には Ruby を使用したので Ruby に移植した上で) 動作させて数えた。前述したように、アドバイザの指摘は必ず誤っているとは限らないので、ここでは「アドバイス相当数」と呼ぶことにする。アドバイザでは最大表示数を 5 件としていたが、ここではそのような上限は設定せず、検出された個数をそのまま使用している。

アドバイザ利用群と非利用群のアドバイス相当数の比較を表 4 に示す。両群の差が統計的に有意かどうかを Welch の t 検定により調べたところ、第 3 回と第 4 回の確認テストが有意であり、また 5 回の確認テスト合計でも有意であった。したがって、アドバイザ利用群は確認テストにおけるアドバイス相当数で数えた「基本的な誤り」の個数がアドバイザ非利用群よりも少ないといえる。

どのようなアドバイスが生成されるかは当然、課題内容 (作成すべきプログラム) によって変化する。表 5 に確認問題の出題内容と、プログラム言語のどのような機能を必要とするかをまとめる。各回の 2 問は p(易しい問題)、q(やや難しい問題) の組になっている。それぞれの問題に対して、想定正解プログラムの行数、メソッド数、代入計算数、ループ数、if 文の数、return の数、使用する配列数、使用する広域変数数を記載した。#2 は単純な代入のみの計算であるのに対し、#3 と#4 が制御構造を組み合わせるため構造の間違いが置きやすく、それに対してアドバイザによる学習が効果

表 5　確認問題の内容

問	テーマ	L	df	cl	lo	if	rt	ar	gv
p02	計算代入	5	1	2			1		
q02	計算代入	8	1	5			1		
p03	条件判断	6	1			1-	2-		
						2	3		
q03	条件判断	8	1			2	2		
p04	配列合計	7	1	2	1		1	1	
q04	配列判断	9-	1	1-	1	1-	2	1	
		13		3		3			
p05	広域変数	7	2	1			1		1
q05	再帰計算	9	1	3		1	2		
p06	2 次元配列	9	1	2	2		1	1	
q06	2 次元配列	11	1	4	2		1	1	

L:行数, df:メソッド定義, cl:計算 (代入), lo:ループ, if:if 文, rt:return, ar:配列, gv:広域変数

的である可能性がある。#5 と#6 は新たに学んだ概念が使えることを確認する問題であるため、制御構造の上での難しさは小さくなっている。

4.3　指摘内容の分布

次に、具体的にどのような誤り (アドバイス) が多いかを検討する。表 6 に、アドバイザと確認テストでどのようなアドバイスが抽出されるかの件数集計を確認テストのアドバイザ非利用群における件数の降順で示した (利用群でも同じ)。「記号」は表 2 で示したものである。かっこ内は件数を人数 (アドバイザと利用群は 54、非利用群は 343) で割った百分率であり、1 人が同じ間違いを複数回犯すこともあるので 100%を超えることもある。なお、この集計ではアドバイザの表示数上限は無視してすべてのアドバイスを集計している。

最も多いのは b の「未代入変数参照」、続いて e の「メソッドの外の return」、c の「メソッドの外の実行文」、t の「end 不足」、d の「return の後の実行文」と続く。これらはアドバイザでも多いが、アドバイザでは t が最も多くなっている。

その後件数に段差があり、p の「余分の end」、q の「値を返すべきなのに return が無い」、r の「パラメタや局所変数の参照がない」が続く。

それ以下は a の「メソッドの中にメソッドがある」、h の「値を返さないはずのメソッドで返値を指定した return」で、構造や仕様としておかしい。最後の u は「グローバル変数の代入がない」

表 6　指摘内容の分布

記号	アドバイザ	利用	非利用
b	50 (92.6%)	140 (259.3%)	1066 (310.8%)
e	36 (66.7%)	131 (242.6%)	909 (265.0%)
c	33 (61.1%)	126 (233.3%)	849 (247.5%)
t	96 (177.8%)	69 (127.8%)	668 (194.8%)
d	19 (35.2%)	49 (90.7%)	540 (157.4%)
p	15 (27.8%)	48 (88.9%)	290 (84.5%)
q	27 (50.0%)	25 (46.3%)	214 (62.4%)
r	43 (79.6%)	30 (55.6%)	201 (58.6%)
a	2 (3.7%)	4 (7.4%)	67 (19.5%)
h	8 (14.8%)	5 (9.3%)	54 (15.7%)
u	0 (0.0%)	1 (1.9%)	4 (1.2%)
n	4 (7.4%)	0 (0.0%)	0 (0.0%)
o	4 (7.4%)	0 (0.0%)	0 (0.0%)
f	2 (3.7%)	0 (0.0%)	0 (0.0%)

で、アドバイザ件数が 0 なのはアドバイザの問題にグローバル変数を扱うものが含めていないためである。

逆に n の「if の中でない else」、o の「if に複数の else」はアドバイザの問題に if-else のやや複雑なものが含まれていたことによると思われる。そして f の「値を返すべき return に式がない」は、選択肢の中に式つきの return と式なしの return が両方含まれているときにだけ犯す間違いであり、確認問題にはそのようなものが含まれていないため 0 件だった。

このように、どのような「基本的な誤り」を犯すかは問題の種別や選択肢の設定より影響を受けるが、全体の傾向として end の不足や余分、メソッドの外に実行文や return を置いてしまうなどの「明らかに動かない」誤りがかなり多くを占めていることは、サポートの必要性を裏付けるものだとも言える。

また、非利用群と比べて利用群の百分率が大きく下がっているものとして b、t、d、q があるが、これらはアドバイザによるアドバイスの効果が大きい可能性がある。

4.4　アドバイスの時系列変化

実際にアドバイス機能を使用した 54 名について、各セッションでどのようなアドバイスが表示され、それに対応したか/しなかったかを図 7 に示す。

表 7　解消されたアドバイスの試行数

試行数	件数
1	170
2	16
3	3
4	3
8	1

1〜54 の数値はそれぞれの学生に対応する (順番づけは任意)。個々の縦線はアドバイスボタンの使用をあらわし、色の濃い線はセッションの開始 (新しい問題に対する最初の使用) をあらわす。英字はアドバイスの記号 (表 2) を表し、横線はアドバイスの表示をあらわす。横線が延びているほど、そのアドバイス (基本的な誤りに対応) が解消されないまま何回もアドバイスボタンを押したことを表す。横線の末尾に×記号があるものは、そのアドバイスが解消されないままセッションが終わったことを表す。

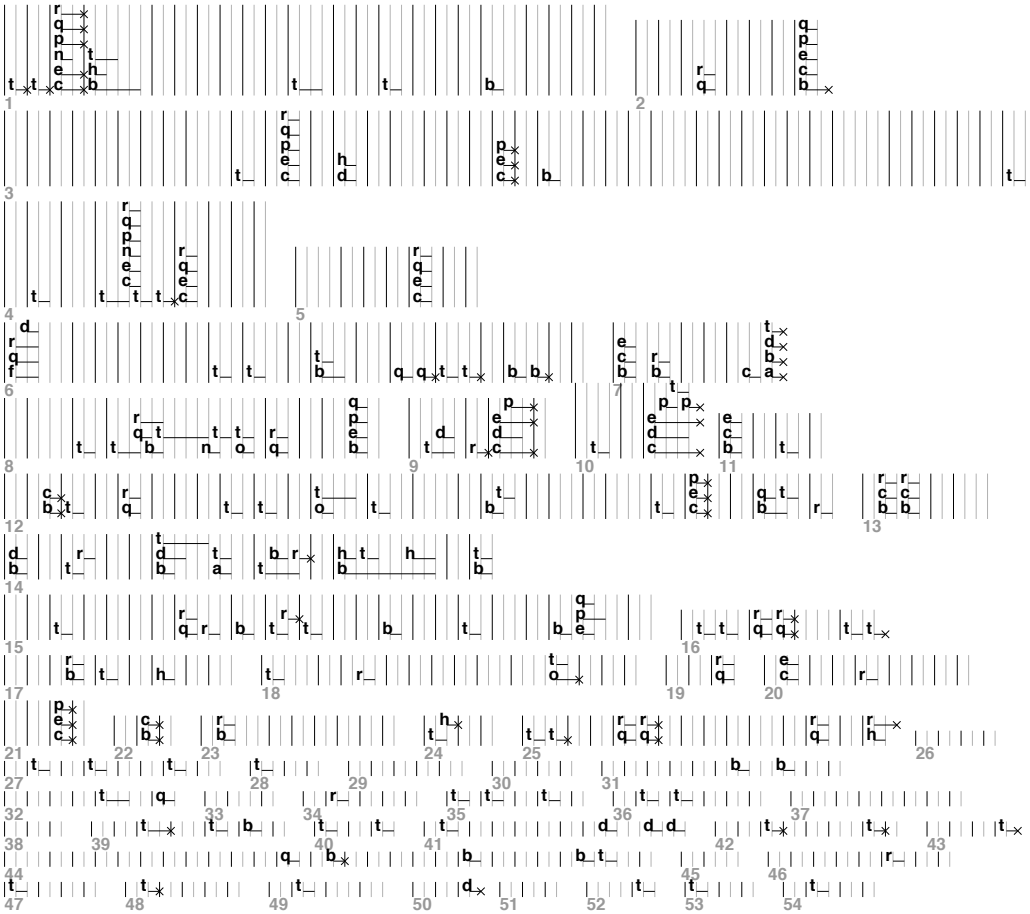
これらを見ると、54 名中アドバイスを全く表示させていない (誤りの検出がない) 者は 8 名、アドバイス総数は 287、うち解消されないままセッションが終わったものは 54(全体の 18.8%)、54 名全体での 1 人あたりアドバイス表示数は 5.3 個であった。解消された 193 件のアドバイスの、解消までのアドバイスボタン押下回数を表 7 に示す。これを見ると、大半のアドバイスは 1 回または 2 回で解消されており、アドバイザが表示するメッセージは十分理解されているものと考えられる。

5.　議論とまとめ

コンパイラのエラーメッセージが初心者に分かりにくいことには定評 (?) があり、このことに関する研究や対する改善の試みは多く存在している [9]。しかし、コンパイラは言語を実装することが目的であることから、間違いでないが間違いの可能性のあるようなものには注力しにくく、最適化の過程で発見した変数の未参照など、本来の仕事に関連して見付かった情報を提供する程度が普通である。

Lisp 系の環境では、Interlisp の DWIM[10] や DrScheme[11] など、親切なメッセージに注力しているものがある。これらの場合、実行時の情報に基づく具体的な助言が得られるという利点がある。

図 7　アドバイスの時系列変化



一方で、Lisp 言語の特性から、静的に検出される誤りにはそれほど注力していない (DrSchem では静的解析に基づくエラー検出も行っている)。

本稿のアドバイザは、プログラミング初心者を対象とし、「end の対応を取る」のようなごく基本的な知識を獲得させることを目的とする点で、これらとは異なっている。また、短冊ベースであることから、スペルミスなど行内の細かい誤りはもともと存在せず、行レベルでの構造の間違いに注力している点が特色となっていると考える。

実際にプログラミング入門科目の中でアドバイザを任意で使用でいるように提供した結果、アドバイザを使用して実際にアドバイスを取得した学生は約 800 人の授業登録者数のうち 176 名、そのうち 3 回以上の使用は 77 名であった。

これらのうちから、授業時の確認問題を「手抜きでなく」受けている集団 54 名を抽出し、アドバイザを使用していない同様の学生と比較したところ、アドバイザが指摘するような誤りについて、有意に少なくなっていることが分かった。実際によく現れる基本的な誤りとしては、代入していない変数を参照する、メソッドの外に実行文があるなどのものが多かった。

また、アドバイザの使用記録を時系列的に調べると、アドバイザを使用している学生は大半の指摘事項について、1 回または 2 回の修正で解消できており、アドバイザが提供している指摘の説明はそれなりに分かりやすく効果的であるものと考えられる。

今後はアドバイザの機能を、確認問題解答時に使用可能するなどの方法で、より多く利用してもらうことにより、学生が速やかに基本的な誤りを

犯さなくなり、より進んだ内容の学習に注力できるようにする方向を考えて行きたい。

謝辞

短冊アドバイザの試用と評価について、電気通信大学共通教育部情報部会 WG のメンバーの協力を頂きました。ここに感謝します。本研究は JSPS 科研費 JP18K02894 の助成を受けたものです。

参考文献

[1] 久野 靖, 情報入試研究会試作問題＃ 001 問題解説, 情報入試研究会, 情報入試フォーラム 2013 資料集, pp. 4-10, 2013.

[2] Parsons D., Haden P., Parsons Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses, Proc. the 8th Australasian Conference on Computing Education2015Volume 52 (ACE '06), pp. 157-163, 2006.

[3] Denny, P., Luxton-Reilly A., Simon B.: Evaluating a New Exam Question: Parsons Problems, Proc. Fourth Intl. Workshop on Computing Education Research 2008 (ICER'08), pp. 113-124, Sept. 2008.

[4] Cheng, N., Harrington, B., The Code Mangler: Evaluating Coding Ability Without Writing Any Code, Proc. SIGCSE'17, pp. 123-128, March 2017.

[5] Yasuichi Nakayama, Yasushi Kuno, Hiroyasu Kakuda, Split-Paper Testing: A Novel Approach to Evaluate Programming Performance, in submission.

[6] 久野 靖, プログラミング入門科目の指針と実践例 (前編), 情報処理, vol. 60, no. 3, pp. 244-247, Feb 2019.

[7] 久野 靖, プログラミング入門科目の指針と実践例 (後編), 情報処理, vol. 60, no. 6, pp. 541-545, May 2019.

[8] 萩原兼一, 大学入試における高校共通教科「情報科」の評価方法改革に関する研究プロジェクト—「思考力・判断力・表現力」を評価する問題の作成方法と CBT による試験実施, 情報処理, vol. 58, no. 9, pp. 840-843, Sep. 2017.

[9] V. Javier Traver, On Compier Error Messages: What They Say and What They Mean, Advances in Human-Computer Interaction, vol. 2010, March 2010.

[10] Warren Teitelman, Larry Masingter, The Interlisp Programming Environment, IEEE Computer, vol. 14, no. 4, pp. 25-33, Apr 1981.

[11] indler R.B., Flanagan C., Flatt M., Krishnamurthi S., Felleisen M. (1997) DrScheme: A pedagogic programming environment for scheme. In: Glaser H., Hartel P., Kuchen H. (eds) Programming Languages: Implementations, Logics, and Programs. PLILP 1997. Lecture Notes in Computer Science, vol 1292. Springer, 1997.