

プログラミング初学者を対象とした概念学習システム

古澤 資 栄^{†1} 高野 辰 之^{†2}
小濱 隆 司^{†3} 宮川 治^{†3}

プログラムが記述されたソースコードにはさまざまな概念が存在する。ソースコードは構文規約に即して記述され、一つもしくは複数の単語の連なりが概念を構成し意味を成している。概念はその名称と複数の性質を持っている。これらの学習はソースコードの読解・記述に無関係であるとはいえない。本研究では、学習者が形成する、概念に対する解釈を把握するシステムを開発し、プログラミングの概念に対する理解度の確認を目指す。

Development of Concept Learning System for Programming Introduction Education

MOTOHIDE FURUSAWA,^{†1} TATSUYUKI TAKANO,^{†2}
TAKASHI KOHAMA^{†3} and OSAMU MIYAKAWA^{†3}

The source code that describes the program there are many concepts. Source code is written according to the syntax rules, which concept form a word or phrase. The concept has a name and properties. And to learn the concepts of source code, may be related to reading and writing source code. In this study, to develop a system to learn the programming concepts in the source code. And to investigate their understanding of programming concepts.

^{†1} 東京電機大学大学院情報環境学研究所

Graduate School of Information Environment, Tokyo Denki University

^{†2} 東京電機大学大学院先端科学技術研究所

Graduate School of Advanced Science and Technology, Tokyo Denki University

^{†3} 東京電機大学情報環境学部

School of Information Environment, Tokyo Denki University

1. はじめに

プログラムのソースコード中にはさまざまな概念が存在する。それらは、一つの単語や複数の単語の連なりによって構成され構文規則と意味規則で定義されている。プログラミング学習者にとって、ソースコードの読解や記述はそれら文法の理解なくして達成は困難である。

教育の分野では知識や技能の獲得を目指す際に技能獲得までのプロセスを階層化し、その問題解決能力を知的技能と定義している¹⁾。階層は、下位から「弁別」「概念」「ルールと原理」そして「問題解決」である。階層の頂点を「問題解決」とし、下位階層から上位階層へと理解を進めなければならない。

ここで、プログラムの読解と記述を問題解決の頂点とする知的技能の階層を定義する。物事の違いを見分けることを意味する弁別は、ソースコード中に存在するそれぞれの単語の識別に相当する。概念の階層は、言語の仕様に沿った規則によって構成される、単語とそれらの連なりがどのような意味を持ちソースコード中においてどのような役割をもつのか理解する工程に対応付けることができる。ルールと原理の階層は、プログラミングでの制御やアルゴリズムに相当する。これは、概念階層で理解した文や式の構成規則から、行われている制御を理解し実行結果を導く。または、それらの制御から目的の結果を導出するまでのプロセスであるアルゴリズムを理解することである。これら下位階層の理解を経てプログラムの読解と記述が可能となり問題解決へと至る。

学習者は概念を習得する際、「何々はどうのようものであるか」という概念的知識の形成を行う。概念とは名称とそれが持つ性質の結合によって定義されている²⁾。学習者は、既知の近似する性質から理解までの工程を導出するか、もしくは、白紙の状態から性質を自身で定義し記憶する。このことから、概念学習は、ある概念の「名称」と「性質」を関連付けて理解することであると定義されている。よって、プログラムの読解と記述を問題解決の頂点とする知的技能の階層における概念学習とは、構文規則などソースコード中に存在する概念の名称とそれらがもつ性質を関連付けて理解することである。

教授者は、授業内で新しい概念がソースコード中に出現するたびにその概念の説明を行い学習者の概念的知識の形成を試みる。そして、学習者がそれぞれの概念的知識の理解度を随時確認し授業を展開する。しかし、通常一対多で進行される授業において、一人一人の学習者の理解を講義で把握することは困難であり、ソースコードなどの成果物からだけでは把握しきれない。

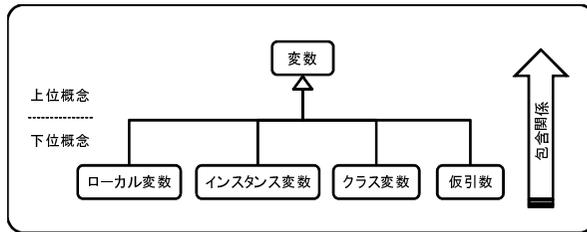


図 1 階層的な包含関係

そこで、本研究では Java 言語を対象としたプログラミングを学習する授業内で使用する概念学習システムを開発し、学習者の概念の理解促進を支援するとともに、解答情報を蓄積し学習者の概念的知識の把握を目指す。本稿では、問題作成に必要な概念のモデル化、システムの仕様と構成、そして、試験運用と解答情報の分析について述べる。

2. 概念のモデル化

概念には性質が近似するものが存在する。性質が近似する概念は、性質的な包含関係をもつ。これらの関係には、オブジェクト指向における継承の Is-a 関係のような、性質の引き継ぎがみられる階層的な包含関係や概念同士の範囲が重なる範囲的な包含関係などがある。本章では、ソースコード中の概念にどのような包含関係が存在するかを示し、概念のモデル化を行う。

2.1 階層的な包含関係

概念には、その概念がもつ性質からみた階層的な関係が存在する。ここで、例として変数を挙げる。変数の概念は値の識別子という性質を持つ。性質的に近似する概念として、ローカル変数、インスタンス変数、クラス変数、仮引数などが存在し、構文規則に従い詳細に分類を行うことができる。仮引数はさらに分類することが可能であるが、変数との包含関係を示すため概括的な概念の分類を行っている。これら 4 種の変数は変数を性質的に包含しており、その関係を階層的に示すことができる。このような概念の包含関係を階層的な包含関係と定義する。

変数における階層的な包含関係を図 1 に示す。この図より、上位概念と下位概念に分類され、下位概念は上位概念と同等の性質を持つ。つまり、下位概念は上位概念が派生した概念であり、上位概念は下位概念の集合であるとなすことが可能である。このように、性質

```
1: public static void main(String[] args) {  
2:     System.out.println(getMessage("Hello"));  
3: }  
4: public static String getMessage(String str) {  
5:     String message = "***" + str + "***";  
6:     return message;  
7: }
```

図 2 範囲的な包含関係

的に Is-a 関係で結ばれているとき、下位概念は上位概念のインスタンスであるとされる³⁾。そして、図 1 右部の矢印は包含の方向を表し、下位概念が上位概念に包含されていることを示している。

2.1.1 階層的な包含関係にある概念の認識

階層的な包含関係にある概念を概括的に理解することに問題は生じない。共通の性質から包含関係にある両者の概念を同一視していたとしても、性質的に近似しているため差分となる性質から構文規則に沿った概念的知識を形成することが可能である。

2.2 範囲的な包含関係

階層的な包含関係では、その概念が持つ性質から階層的に概念の構造を示した。ここでは、ソースコード中の概念が存在する範囲による包含関係について述べる。

図 2 は、ソースコードからメソッドの宣言部を抜粋したものである。5 行目は変数宣言および初期化構文を表している。変数宣言部は変数の型である「String」と変数名である「message」で構成されている。つまり、変数宣言という概念は、変数の型と変数名の概念を範囲的に包含している。このような概念の包含関係を範囲的な包含関係と定義する。範囲的な包含関係は、一つの変数が複数の概念で構成されることを意味し、概念の入れ子構造を表している。

図 2 の 6 行目「message」は 5 行目で宣言された変数であると同時に、String 型の値を返却するメソッドである「getMessage」の返却値である。よって 6 行目の「message」は、変数名と返却値という 2 つの概念を含んでいる。5 行目の範囲的な包含関係とは異なり、概念がある概念の構成要素になっているわけではない。

return 文である 6 行目の処理によって呼び出し元に返却されるのは、式もしくは、式が評価された値である。そして、メソッドである「getMessage」の呼び出し元が受け取るのは値である。これらの値または式には「返却する・返却された」という性質が含まれており

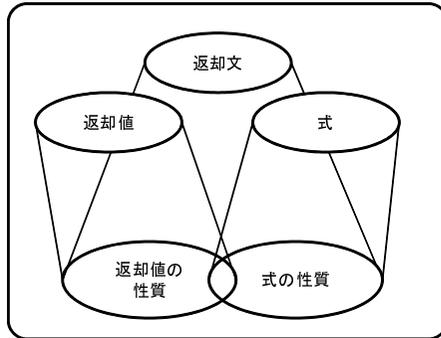


図 3 返却値と式の関係

返却値と定義される。

返却値と式の関係を図 3 に示す。式は、return 文中に存在するとき返却値として扱われる。これは、式が return 文中に存在することで役割が変化しており、新たな異なる概念の構成要素となっているといえる³⁾。また、呼び出し元に返却される値もメソッド呼び出し部と値の間に同等の関係を持っている。よって、返却値の概念は、return 文中の式や、呼び出し元のメソッド呼び出し部などの概念を範囲的に包含しているといえる。

2.3 その他の概念の包含関係

ソースコード中の概念には、言語仕様によって定められた構文規約上の概念のほかに、プログラミング学習などで用いられる学習上の概念がある。これらは、概念の性質から概念を抽出したものや概念が所属する場所などから新たに概念が付加された概念である。

Java 言語には反復的な制御を行う構文として for 制御構文と while 制御構文がある。二つの構文を使用し、同様の動作を実装することは可能であるが、行っている制御は異なり、この二つの制御構文は全く別の概念である。しかし、二つは「繰り返しを行う制御構文」という性質を持ち、共通の性質から「繰り返し」という概念を形成している。

繰り返しという概念は、for 制御構文と while 制御構文を範囲的・意味的に包含しており、性質を概念として抽出した「繰り返し」という概念は包含している二つの概念を抽象化した概念であるといえる。

また、条件分岐を司る制御構文として、if 制御構文と switch 制御構文が存在する。これらは、繰り返しと同様に、「条件分岐を行う制御構文」という共通の性質から、抽象化された概念である「条件分岐」の形成が可能である。

2.4 概念の学習

ソースコード中の概念は、その概念を形成している範囲と名称、そして、性質の結合からなる。階層的な包含関係をもつ概念の継承的な関係を示した。概念は上位概念を持つことがあり、下位概念は上位概念の性質をすべて有している。範囲的な包含関係をもつ概念では、ソースコード中の一つの範囲に対して複数の概念が存在する場合があります。それらが持つ性質や範囲から異なる概念が付加される。よって、ソースコード中におけるプログラミングの概念は範囲に属しさまざまな性質を包含している。

これらから、プログラムの読解と記述を問題解決の頂点とする知的技能の階層における概念学習は、ソースコード中の範囲と対応づけられた概念がどのような概念であるかを思考し、対象となる概念が持つ名称や性質を範囲に対して関連付けることである。

3. 概念学習システムの仕様

前章において、プログラムの読解と記述を問題解決の頂点とする知的技能の階層における概念学習に必要な概念のモデル化を行なった。本章では、それらの概念理解を支援し、その理解度を確認するシステムの仕様について述べる。

3.1 目的

学習は、知識の形成・確認・修正を反復して行う。ソースコードにおけるプログラミングの概念でも同様に、授業や自主学习から概念の名称と性質の対応付けを行い、概念的知識を形成する。

概念学習システムは、概念的知識の確認を主眼とする。学習者にはソースコードが提示され、概念が存在する範囲に対して名称を解答する。解答後、システムによって採点され学習者に正誤が通知される。教授者は、出題されたソースコードに対して概念の解説を行うことで概念的知識の修正、または、形成を目指す。そして、解答情報から学習者の概念形成を確認し授業へのフィードバックを行う。

3.2 出題対象言語

出題対象のプログラミング言語について、本システムは Java 言語を対象としている。しかし、システムは、概念(モデル)部、ソースコードの構文解析部、ユーザインターフェース部で分離されており、対象言語の変更は、言語ごとに定義される概念部とソースコードの解析部を入れ替えることによって可能である。

3.3 教育者側の仕様

本システムは、教授者と学習者にそれぞれ、異なるクライアントアプリケーションを提供

表 1 概念の範囲

単語 1	単語 2	単語 3	単語 4	単語 5	概念の名称
String					変数の型
	message				変数名
		=			代入演算子 (初期化構文の一部)
			"Hello World !"		変数の値
				;	区切り子
String	message				変数宣言
	message	=	"Hello World !"		代入文
String	message	=	"Hello World !"	;	変数宣言および初期化構文

する。教授者のクライアントアプリケーションでは問題の作成を行う。図 4 に教授者側クライアントアプリケーションのイメージを示す。ここでは、教授者が問題を作成する工程から、問題作成における仕様を述べる。

3.3.1 出題概念の選択と対象ソースコードの選択

問題作成では、出題したい概念の選択を行う。図 4(1) の概念一覧から出題したい概念にチェックを入力し、出題対象のソースコードをファイル選択欄から選択する。

本システムにおいて、一つの単語は一つ概念にのみ属する。表 1 は変数宣言および初期化構文とそこに含まれる概念の範囲を示したものである。変数宣言および初期化構文を構成する概念として、変数宣言部と代入文部がある。さらに、変数宣言部は変数の型と変数名を、代入文部は変数名、代入演算子、そして、変数の値を範囲的に包含している。

一つの単語は一つ概念にのみ属するため変数宣言および初期化構文を出題する場合、変数宣言部や代入文部、または、変数の型など範囲的な包含関係により、範囲が重複する概念は出題することができない。これは、学生に提示される、解答欄となるソースコード中の範囲が増加し、難易度が上昇しすぎることを懸念したためである。

図 4(1) の概念一覧に分木的な構造が存在するのは、範囲的な包含関係を表現し、出題する概念の選択を制約するためである。ある概念を選択した場合、その概念の下部にある構造的にみたま下層の概念を選択することはできない。例えば、「クラス宣言」を選択した場合は、範囲的に重複する下層の概念を選択することができない。

また、階層的な包含関係を持つ概念について、上位概念と下位概念を同時に問題作成することはできない。上位概念と下位概念の間には Is-a 関係が存在するため、混乱を招き、円滑な解答を阻害する恐れがある。例えば、階層的な包含関係上の上位概念である「変数」に関する概念を出題するとき、下位概念である「インスタンス変数」や「ローカル変数」などの概

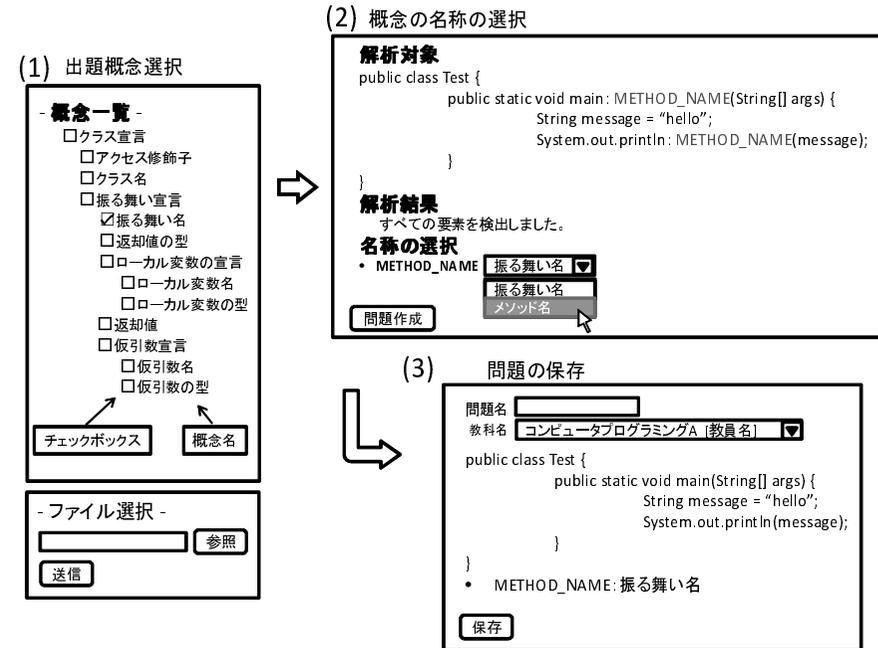


図 4 教授者側クライアントアプリケーションイメージ図

念を出題することはできない。

なお、解析対象のソースコードから出題したい概念が検出できなかった場合は解析の失敗を問題作成者に提示する。このとき、問題作成者は、解析対象となるソースコードの変更、または、出題する概念の再選択を行わなければならない。

3.3.2 概念の名称の選択

概念とは名称とそれが持つ性質の結合によって定義されている。概念の名称によって、概念を一意に識別することは可能であるが、名称は概念に対して一つであるとは限らない。そのため、出題時に学習者に提示される選択肢名を決定するために、概念の名称が複数ある場合は名称の候補から選択しなければならない。

図 4(2) は、出題概念として振る舞い名を選択し、システムに提示したソースコードから解析を行った例である。例では、出題する概念として振る舞い名を選択し解析を行っている。

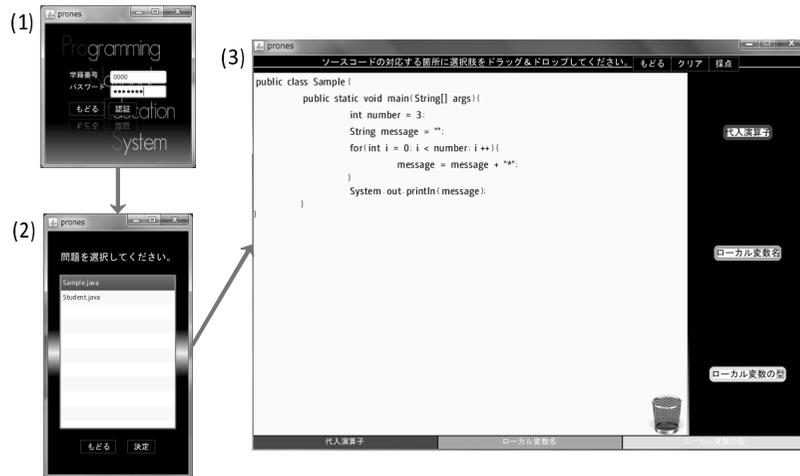


図 5 学習者側クライアントアプリケーション

ここでは、名称が複数存在する概念としてメソッド名があり、提示された「メソッド名」, 「振る舞い名」の候補から一つを選択しなければならない。選択終了後、問題作成ボタンを押し対象授業の選択に移行する。

3.3.3 出題対象の授業と問題名の入力

作成される問題は、授業ごとに管理されている。教授者は問題を作成する際、どの授業内で本システムを用いた演習を行うかを選択する。図 4(3) は、授業の選択を行う画面である。教授者は、教科名と担当教授者名が表示されているセレクトボックス内の候補から該当する授業を選択する。教授者は、以上の作業を終了し「保存」ボタンを押すことで問題の作成を完了する。

3.4 学習者側の仕様

学習者が利用する解答用のクライアントアプリケーションについて、解答までの過程から仕様を述べる。図 5 に学習者側クライアントアプリケーションの画面を示す。

3.4.1 認証

学生は、本システムの解答インターフェースを利用する際に認証を行う。図 5(1) は、実際に学生が認証を行う画面である。学生は、学籍番号とパスワードをシステムに入力し認証を行う。



図 6 ドラッグアンドドロップによる解答操作

3.4.2 問題の選択

前述したように、問題は授業ごとに管理されている。学生は、教授者が指定した URL にアクセスし、クライアントアプリケーションである解答用のインターフェースを起動させる。既定の URL へアクセスすることで学習者が受講中の授業に対して作成された問題を取得することができる。認証を終えた学生は、図 5(2) のインターフェースの問題一覧から解答すべき問題を選択し解答を開始する。

3.4.3 問題への解答

図 5(3) の解答画面は、解答欄となるソースコードと選択肢、そして、各概念と対応する色と選択肢の個数を示す判別部で構成される。図 6 に解答操作を示す。学生は、ソースコード中の箇所に対応する概念の名称となる選択肢を右部の選択肢欄からソースコードにドラッグアンドドロップすることで解答を行う。

選択肢の個数には制限があり、その概念がソースコード中に存在する個数分だけ存在す

る。選択肢の個数は、各概念の判例に示され、選択肢をソースコードに対応させるたびに減少し、残り個数に従ってカウンタが増減する。選択肢の個数に関して、制限を取り外すことが可能である。その場合、学習者は選択肢を任意の箇所にもいくつでも解答することができ、判例部の選択肢の個数は表示されない。

解答動作として、一度対応させた箇所から別のソースコード中の箇所に解答を移すことも可能である。解答の取り消しは、ソースコード右下のゴミ箱に取り消したい箇所の解答をドラッグアンドドロップすることで行うことが可能であり、解答の取り消しを行うと、解答時とは対称に判例の選択肢の個数が増加する。

解答を終えた学生は、採点ボタンを押し採点を行う。対応させた選択肢がその範囲に含まれる概念と同値でなければ誤りを通知し、対応させた選択肢とその範囲に含まれる概念に上位概念が存在するとき、それぞれの上位概念が同値であることを通知している。なお、自身が解答した箇所の正誤は通知されるが模範解答は示されない。

また、学生は何度でも解答することが可能である。一度解答を終えた場合、「クリア」ボタンで自身が行った解答をすべてリセットし、初めから解答を行うか、もしくは、リセットをせずに誤答が示されている状態から修正し、再度採点することも可能である。

「もどる」ボタンでは、前画面への遷移が可能である。提示される問題は、画面遷移毎に再読み込みするため、解答インターフェース起動後の問題作成も可能である。

4. システム構成

本章は、教授者と学習者にそれぞれ提供されるクライアントアプリケーションであるフロントエンド部、そして、問題・解答情報の管理を担うバックエンド部についての構成を述べる。

バックエンドとフロントエンドの構成を図7に示し、それぞれの詳細について述べる。

4.1 バックエンド

バックエンドでは、クライアント間で利用されるモデルとソースコード解析の機能を提供している。モデルは各クライアントがやり取りをするデータモデルを定義し、それらを利用した問題情報や解答情報などがデータベースに保持される。また、これらを SOAP(Simple Object Access Protocol) を利用したウェブサービスとして提供しており、他システムとの連携が容易となっている。

4.1.1 ソースコード解析

ソースコード解析には Java の構文解析器を利用し、ソースコード上の範囲情報から概念

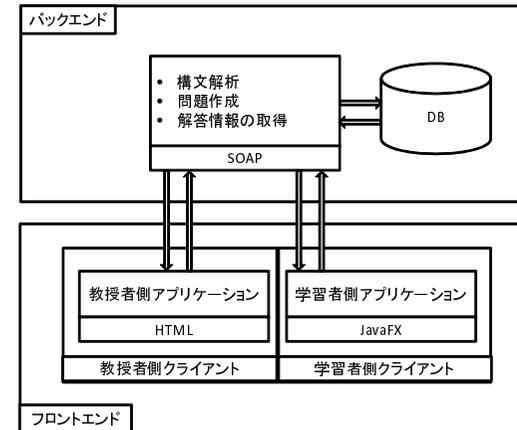


図7 システム構成

を生成している。解析対象のソースコードと出題したい概念を入力として与えることで、範囲に対応付けられた概念や解析の成否などの解析結果情報が提供される。解析には、出題対象となるソースコードのみを使用する。コンパイルをせずに解析を行うため、構文規約に沿ったソースコードであるなら解析を行うことが可能である。

4.1.2 解答情報の取得

教授者は、学習者の解答情報から算出される各概念の理解度や誤答情報などを取得することが可能である。また、学習者や授業、問題ごとにそれらの情報を絞り込み、必要な情報を選定する。

4.2 フロントエンド

教授者と学習者が利用するフロントエンドについて述べる。教授者と学習者には異なるクライアントアプリケーションが提供され、教授者は問題作成、学習者は解答を行う。

4.2.1 教授者

教授者は、ブラウザを利用して問題の作成を行う。前述した、ウェブサービスによって提供される機能から、問題作成や解答情報閲覧を行うためのウェブページを生成する。

4.2.2 学習者

学習者のクライアントでは、ドラッグアンドドロップによる直観的な解答動作やグラフィカルな画面遷移を実現するためRIA(Rich Internet Application)であるJavaFX Scriptに

```
public class Sample {  
    public static void main(String[] args) {  
        int number = 3;  
        String message = "";  
        for(int i = 0; i < number; i++) {  
            message = message + "i";  
        }  
        System.out.println(message);  
    }  
}
```

図 8 Sample.java

よって実装を行った。教授者が指定した URL にアクセスすることで、アプリケーションがダウンロードされ Java Web Start によって動作を開始する。

5. 試験運用

本システムを実際の授業に適応し試験的に運用を行った。本章では、運用の対象となった授業と出題の内容など、運用の詳細について述べる。

5.1 対象

対象となった授業は、コンピュータプログラミング A という講義である。出題の対象となった学生は 48 名で、2011 年 10 月 26 日の講義内で実施した。

本学部には、プログラミングを扱う講義としてコンピュータプログラミング A、コンピュータプログラミング B が存在し、Java 言語を対象言語としている。コンピュータプログラミング A はプログラミング初学者を対象とし、Java 言語を使用して手続型プログラミングの考え方を学習する。コンピュータプログラミング B では、コンピュータプログラミング A を履修した学生を対象とし、Java 言語を使用したオブジェクト指向プログラミングの基本的な概念を学習する。

5.2 出題

出題対象となったソースコードは図 8 に示す「Sample.java」である。学生は解答すべき概念として「ローカル変数名」を提示され解答を行った。

図 8 のソースコード中には、3 行目の「number」、4 行目の「message」、for 制御文中の「i」が変数として定義され、ローカル変数名は 9 箇所存在する。学生は、システムによって

表 2 誤答パターン一覧

誤答箇所の概念	回数
ローカル変数の型 (無規定)	24
値	12
返却値の型	20
加算演算子	2
仮引数名	1
仮引数の型	1

通知される誤答箇所がなくなるまで解答を行った。

6. 運用結果

本章では、試験運用によって得られた解答情報から正答率や誤答のパターン分析、誤答の一貫性による分析を行い、指導に有益となりえる情報を導き出す。なお、解答情報は初回解答のみ利用する。

6.1 正答率

試験運用で出題した変数名の正答率は、67.5%となった。また、一つも誤答がなかった学生は 48 名中 19 名だった。

正答率 [%] は以下の式によって計算した。

$$\text{正答率} = \frac{\text{正解数の総和}}{\text{解答箇所数} \times \text{解答者数}} \times 100$$

6.2 誤答のパターン

ローカル変数に対する誤答のパターンと回数を表 2 に示す。誤答箇所の概念の無規定は、構文解析上、概念を規定していない箇所である。本システムの仕様では、概念を規定していない箇所に対しても解答を行うことが可能となっている。なお、無回答による誤答は表から排除している。

6.3 誤答情報の一貫性による分析

誤答の一貫性とは、ひとつの解答中に認識の一貫性が存在することを指す。例えば、図 8 のソースコードに対して解答をするとき、3 行目の変数の型である「int」に対して変数名という解答を行ったとする。ここで、同様に 4 行目の変数の型である「String」に対しても変数名という解答を行った場合、認識に一貫性が存在していると言える。しかし、同等の概念をもつ範囲に対して、解答毎に異なる概念を解答するような一貫性のない誤答が存在する。

表 3 誤答の分類と人数

	無回答による誤答あり	無回答による誤答なし	計
一貫性のある誤答	6	2	8
一貫性のない誤答	20	1	21
計	26	3	29

一貫性のない誤答が示唆するのは、誤った形で概念的知識が形成されているか、もしくは、概念的知識自体が形成されていないかである。誤った形で概念的知識が形成されている場合、名称と性質の対応付けに問題が存在する可能性があり、再度、名称と性質を関連付けなければならない。また、概念的知識自体が形成されていない場合、一から知識の構築を行わなければならない。

誤答の分析として、学習者ごとの解答から一貫性のない誤答を検出し、概念的知識の形成の有無を確認する。そして、一貫性のある誤答情報を蓄積することで、学習者が犯しやすい誤認のパターンを調査する。

6.3.1 分析結果

誤答を分類し、集計結果を表 3 に示し、各項目について述べる。一貫性のある誤答は、学習者が変数名と解答した箇所の概念に一貫性が存在するものである。そして、一貫性のない誤答は、変数名の正解数に関わらず、解答箇所の概念に一貫性がないものである。また、無回答による誤答は、一か所以上変数名の箇所に解答したものの、その他の変数名の箇所は無回答だった誤答を示す。つまり、無回答による誤答ありは、解答すべき概念の正解数が、1~8 個である場合、無回答による誤答なしは、変数の正解数が、0 個もしくは、解答すべき概念の出題のすべてに正解した場合である。

次に、ソースコード上の解答箇所ごとの誤答数を算出した。出題したソースコードの解答箇所の番号を図 9 に示す。解答箇所別の誤答数を図 10 に示す。誤答が最も多かったのは、for 制御文中の「i」宣言部、最も少なかったのは「number」宣言部となった。

7. 考 察

本章は、試験運用結果についての考察を述べる。無回答以外の総誤答数が少なかったため、誤認しやすい概念を示すような誤答の兆候はみられなかった。そして、総誤答人数に対して一貫性のある誤答が極端に少ないことから、誤答を犯した学習者の大半はソースコード上の同等の性質を持つものに対しての同一視ができていないと推察される。

```
public class Sample {
    public static void main(String[] args) {
        int ① number = 3;
        String ② message = "";
        for(int ③ i = 0; ④ i < ⑤ number; ⑥ i++) {
            ⑦ message = ⑧ message + "**";
        }
        System.out.println(⑨ message);
    }
}
```

図 9 解答箇所

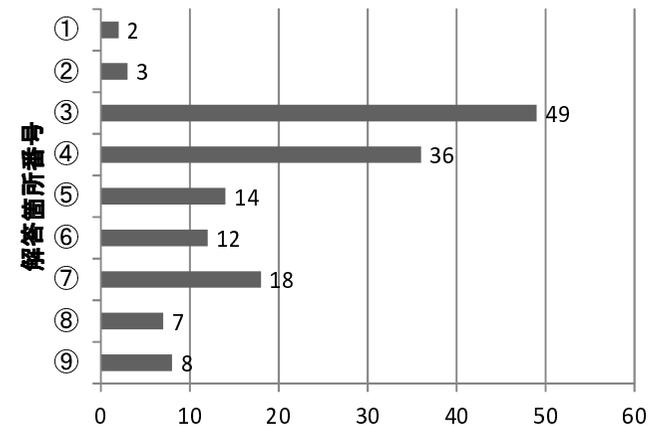


図 10 箇所別誤答個数

図 10 の解答箇所別の誤答数から、変数の宣言部に関しては理解されていることがわかる。これに対して、for 制御文中の構造が深い部分での誤答が多くみられる。また、for 制御文と変数の性質に関する理解が曖昧であることが推察される。

for 制御文などの制御構造を読解する際、必然的に考慮しなければならない情報量が増加し複雑になる⁴⁾。したがって、ソースコード中から変数名を見つけることが困難になり、無回答による誤答を引き起こしていると推察される。

また、一貫性のない誤答を分析すると、そのほとんどが無回答による誤答をしていた。こ

れは、理解が曖昧であり、変数名の概念に関して、他概念との誤認があると考えられる。

8. 関連研究

Richard Bornat は、一貫性とプログラミング適正の関連性を調査している⁵⁾。この調査では、変数間の値のやり取りを正確に理解できているかを学習者に課題し、得られた解答の一貫性からプログラミングの適性を推定している。本システムでは、代入文の制御を理解する前段階である変数の識別から、実際に代入が行われている代入文の識別のみならず、さまざまな概念の課題が可能である。したがって、本システムを利用し、それぞれの理解度と解答一貫性を調査することでプログラミングの読解能力を類推することが可能である。

東野らは、再帰アルゴリズムの認知構造と教育方針について調査している⁶⁾。そして、林は再帰呼び出しを含む処理の難しさについて調査している⁷⁾。いずれも、再帰というアルゴリズムについての学習者の認識の構造について着目しており、本稿で定義したプログラムの読解と記述を問題解決の頂点とする知的技能の階層における「ルールと原理」層にあたる。本研究は「ルールと原理」層の下層に位置する「概念」層に着目しており、より初学段階の学習者に焦点をあてているといえる。

Pauli Byckling らは、プログラミング初学者のプログラム作成手順を調査し、変数の役割をアニメーションを用いて学習する手法を提案している⁸⁾。本研究では、システムによる演習を通して学習者の概念的知識の形成を目指すとともに、学習者の理解状況を把握することを主眼としている。そして、演習を重ねることで、授業の進度や学習者に合わせた概念学習を提供することが可能である。

9. まとめ

本稿では、プログラミングに使用するソースコード中の概念についての理解の促進と理解度の把握をするために、プログラミング初学者を対象とした概念学習システムを提案した。ソースコード中の概念のモデル化と本システムの概要を述べ、試験運用の結果から解答情報の分析を行った。

その結果、誤答情報を分類することで、概念の詳細な理解状況の把握が可能となった。ここから、各学習者の概念的知識に関して授業へさまざまなフィードバックができると考えられる。誤答の一貫性を調査した結果からは、半数近い学習者の誤答に一貫性がなく、概念に対する理解がなされていないことが確認された。このような誤答をした学習者や概念への理解度が低い学習者に適切な指導を行っていきたい。

また、概念的知識の形成のプログラミング教育における有用性を検証する必要がある。概念的知識の有無によってプログラミング記述能力を推しはるのは困難である。ソースコードの記述では、行われているすべての処理を完全に理解していなくても可能である。これは、学習者がコンパイルエラーなどの間違いを繰り返し、実行結果からソースコードを完成させることも不可能ではないからである。そして、概念的知識の形成がソースコードなどの成果物からでは把握しきれない理由でもある。

しかし、プログラムの読解に関して、ソースコードから実行結果を導出するためには、前述したプログラムの読解と記述を問題解決の頂点とする知的技能の階層における弁別・概念層の習得が必須である。一行一行がどのような動作をし、どのように値がやり取りされているのかを理解するためには、ソースコード上の同等の性質を持つものに対して同一視を行わなければならない。よって、今後の運用から本システムによる概念的知識の形成とプログラムの読解能力について関連が存在するかを厳密に調査し情報の有用性を確認していきたい。

参考文献

- 1) R.M. ガニエ, W.W. ウェイジャー, K.C. ゴラス, J.M. ケラー: インストラクショナルデザインの原理, 北大路書房 (2007). 鈴木克明, 岩崎望 監訳.
- 2) 安西祐一郎, 市川伸一, 外山敬介, 川人光男, 橋田浩一: 岩波講座 認知科学 2 脳と心のモデル, 岩波書店 (1994).
- 3) 砂川英一, 古崎晃司, 來村徳信, 溝口理一郎: コンテキスト依存性に基づくロール概念組織化の枠組み, 人工知能学会論文誌, Vol.20, pp.461-472 (2005-11-01).
- 4) 桑原恒夫: 例題中心学習における教材の知識構造の複雑さと理解の困難さとの関係, 電子情報通信学会論文誌, Vol.80, No.11, pp.3039-3047 (1997-11-25).
- 5) Bornat, R., Dehnadi, S. and Simon: Mental models, consistency and programming aptitude, *Proceedings of the tenth conference on Australasian computing education - Volume 78*, ACE '08, pp.53-61 (2008).
- 6) 東野勝治, 白田昭司, 葎谷安正: 再帰アルゴリズムの認知構造と教育方略, 高等専門学校教育と研究: 日本高専学会誌, Vol.3, No.3, pp.30-35 (1998-07-31).
- 7) 林 創: 再起呼び出しを含む手続きの処理の難しさ, 認知科学, Vol.6, No.4, pp.389-405 (1999-12-01).
- 8) Byckling, P. and Sajaniemi, J.: Roles of variables and programming skills improvement, *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, SIGCSE '06, pp.413-417 (2006).