# 問題テンプレートを用いたプログラミング学習における空欄補充問題の自動生成に関する研究

2009SE060 堀井和稀　2009SE239 西條広亮
指導教員：蜂巣吉成

## 1　はじめに

プログラミング言語の学習は，プログラミングの演習問題を繰り返し複数解き多くのプログラムに触れることで学習効果が上がると期待できる．問題の一種として空欄補充問題が挙げられる．文献 [1] では学習意図をパターンで記述し，ソースプログラムに適用し，空欄補充問題を自動生成する方法を提案している．文献 [2] では空欄補充問題の自動出題・採点システムが提案されており，出題者や採点者の負担を軽減することができる．しかし，同じプログラムに対する空欄補充問題を繰り返し出題しても，学習者がプログラムを覚えてしまう恐れがあるので，学習して欲しい意図に応じて，その部分を様々に変えたプログラムを多く用意することが望ましい．

問題として適切なプログラムを多く作成するには新規の問題を作成する，もしくは既存の問題を手作業により修正する方法が考えられるが，これらの方法は問題作成者の負担が大きい．出題する際に必要となるプログラムは学習意図の部分が異なるが，他の部分は共通のものも多い．エディタで正規表現などを用いて書き換えることも考えられるが，適切な正規表現を記述することは難しく，置換漏れや意図しない置換が行われていないか確認することも必要で手間がかかる．空欄補充問題として出題するには，プログラムだけでなく，そのプログラムに対する説明文も必要であるが，プログラムと説明文を書き換え，整合性を保つことができているか確かめるには手間がかかる．

本研究では空欄補充問題のためのプログラムと問題文を自動で生成することを目的とする．空欄補充問題に用いられるプログラムは学習意図の部分が異なり，他の部分は共通であることが多いことに着目して，共通部分をテンプレートとして，異なる箇所をパラメータとして記述する方法を提案する．テンプレート部分はプログラムと説明文から構成し，パラメータ部分にはプログラムの記述と自然言語による説明文をひとまとめにして記述する．パラメータをテンプレートに埋め込む際は，1 対 1 の単純な文字列置換ではなく，不定個数のパラメータを適切に埋め込めるような制御構造も記述可能にした．

## 2　関連研究

空欄補充問題の問題文およびプログラムの自動生成に関する研究を紹介し，本研究に既存研究の手法が応用可能かどうかを議論する．

### 2.1　プログラミング学習のための QA サイクル:受講者の習得度に応じた問題自動提示メカニズム

文献 [3] では，EDML(Exercise Design Markup Language) による QA サイクル実行システムを提案している．EDML は同じような難易度で内容の異なる複数の問題の自動生成と受講者の習得度に応じた繰り返し学習のための答案評価環境と習得度に合わせた問題選択の指定を 1 つのテキストとして記述する言語である．教師は問題文やプログラムの一部を「チャンク」として指定し，チャンクに対する置換可能チャンクを設定する．置換可能チャンクとは，問題の難易度が変わらず，他のチャンクと組み合わせたときに，正常なプログラムとして動作する別のチャンクである．チャンクを置換することで派生問題を生成することができる．しかし，置換可能チャンクは元のチャンクの文字列置換でしかないので，必要とする変数の個数が異なる問題や，else if 部分の個数が異なる問題などは生成できない．

### 2.2　eRuby

eRuby[4] は，テキスト内に Ruby を用いた文を挿入することによって文書の一部を書き換えることを可能としている．テキスト内に制御構文を挿入し，書き換える点が本研究と似ている点である．if 文の条件式の数が異なり，与えられた条件式の数に応じて if-else 文を自動生成することは可能であると思われるが，その場合には Ruby を用いた制御構文が埋め込まれたテキストが読みづらくなることが考えられる．また，プログラミング問題の自動生成に用いるテンプレートを eRuby を用いて記述しようとすると，Ruby によって記述された部分が長くなり，テンプレートが読みづらくなってしまうと考えられる．これは，Ruby がテンプレートを記述するための言語ではなく，汎用的なプログラミング言語だからである．

### 2.3　JSP

JSP(JavaServer Pages)[5] は，HTML 内に JSP タグを埋め込み，サーバで書き換えた HTML をクライアントに返す．タグを埋め込んだ文字列をプログラムによって書き換えて出力する点が本研究と似ている．問題点としては eRuby 同様の点と，HTML ファイルをサーバ内で書き換えてブラウザで表示する仕様上，出力形式が HTML に限られる点が挙げられる．

### 2.4　考察

上記の関連研究では，問題文とプログラムを関連づけて書き換える場合や条件分岐の数が異なる場合などに書き換えが困難であったり，テキストが複雑になり読みづらくなってしまうことが考えられるので，本研究ではプログラミング学習のためのプログラム生成に特化したテンプレートを提案する．

## 3 演習問題の分析

### 3.1 テンプレート化可能な問題数

南山大学情報理工学部の 2009 年度，2010 年度，2011 年度のプログラミング基礎実習のレポート課題を調査したところ，34%の問題が他の問題と共通の部分が多く，テンプレートで表現可能であることが分かった．これらの問題は年度や課題毎に，その課題で学ぶべき項目 (学習意図) の箇所の式や値などが異なっていた．

### 3.2 分析結果

テンプレート化できると判断した問題の詳細を分析すると，次のことがわかった．

1. 問題によって変数の名前や個数，条件判定が異なる
   例えば，教科の成績判定の問題では，国語，英語，数学の 3 教科から成績を判定する問題と，3 教科に理科，社会を加えて 5 教科で判定する問題がある．成績の基準も各教科の合計点だけのものと，合計点と各教科の点数で判定するものがあり，成績も 2 段階 (合否のみ) や 3 段階 (A，B，C) に分けられる．
2. 繰り返しの方法が異なる
   アスタリスクで左上や左下が直角の三角形などの図形を描く問題では，繰り返しで記述できるが，繰り返しの方法が n から 1 まで，1 から n までと異なる．
3. 実行されない文がある
   アスタリスクで右下や左下が直角な三角形などの図形を出力する問題では，右下が直角な三角形の場合は各行のアスタリスクの左側に空白を出力するが，左下が直角な三角形の場合は空白を出力しない．アスタリスクを出力する方法は，右下が直角でも左下が直角でも同じである．

### 3.3 パラメータに設定すべき箇所

3.2 節の分析結果から，プログラミングを学習していく上で重要な箇所のうち，テンプレートを作成していく上でパラメータに設定すると効果的な問題を作成できると思われる箇所を単元ごとに分けて挙げていく．

入出力の学習意図としては，自然言語や変数の入出力方法が挙げられる．

if 文，switch 文の学習意図としては，if-else if-else，switch の構造や条件式の記述などが挙げられる．これらの意図に応じて else if，case の分岐数が異なる問題や，条件式の記述が異なる問題などが考えられる．例えば，科目の点数を入力してその合計点などで合否判定や A+,A,B,C,F などの成績判定を行う問題などが挙げられる．これらの問題ではプログラムの大きな流れ (点数の入力，合計点の計算，合計点による条件分岐と結果の出力) は共通で，分岐数や条件式，出力する文字列などが異なる．

配列の学習意図としては，配列の宣言方法，添字の理解，配列を操作する際のポインタ変数の増減が挙げられる．

構造体の学習意図としては，構造体の宣言方法，参照方法が挙げられる．

関数の学習意図としては，関数の作成方法，返り値の理解，呼び出しかたが挙げられる．

以上の箇所をテンプレート内でパラメータとして記述することにより類似問題の自動生成が可能となり，異なるパラメータを増やすことによって学習者にとって効果的なプログラミング問題を用意する手間と入力ミスを減らすことができると考えられる．

## 4 問題テンプレートの設計

### 4.1 テンプレート全体の設計

3 節の分析結果から，テンプレートを次のように XML を用いて設計した．XML を用いたのは，データの表現形式として広く普及し，テンプレート全体を問題文部，プログラム部，パラメータ部を同じファイル内にわかりやすく記述するためである．

テンプレートは全体を<prog-template></prog-template>のタグで記述し，その中に問題文部，プログラム部，パラメータ部の 3 つを記述する．問題文部では問題文を自然言語で記述し，<prog-explanation></prog-explanation>タグで囲む．プログラム部ではプログラムを記述し，<program></program>タグで囲む．本研究ではプログラムは C 言語で記述する．パラメータ部は<parameter></parameter>タグで囲み，問題文部とプログラム部でパラメータとなっている部分に実際に与える値を記述する．

パラメータとは問題毎に異なっている部分のことであり，これを変更することにより 1 つのテンプレートから複数の問題を生成することが可能である．

問題文部とプログラム部では，生成したい各問題によって異なる部分をパラメータ変数によって記述し，各問題に共通する部分をパラメータ変数ではなく自然言語もしくは C 言語の文法にしたがった文を記述する．

問題文とプログラムは 1 対 1 で対応しているので，同じファイル内に記述することによって管理しやすくなる．問題文部とプログラム部ではテンプレート化する問題の共通部分を記述し，異なる部分はパラメータ変数や制御命令で記述する．これらを XML を用いて記述することもできるが，タグが増えて可読性が低下したり，記述の手間が増えたりするので PHP や eRuby などを参考にしてパラメータ変数は “${変数名}” として，制御命令は “<% %>” で囲んで記述することにした．

パラメータ部には問題文部，プログラム部で使われる全パラメータの実際の値を “${変数名}＝値;” として指定する．テンプレートにはパラメータ部を複数記述することができ，1 つのパラメータ部から 1 つの問題が生成される．

### 4.2 変数の設計

3 節の分析により，類似問題には共通している部分と異なる部分があることがわかったので，問題によって異なる部分はパラメータを用いて書き換える必要があり，パラメータ変数を用いる．パラメータ変数はアルファベットと数字，ハイフンで構成し，“${変数名}” として記述する．なお，パラメータ変数の先頭はアルファベットと

する．パラメータ変数に格納するデータは，プログラムにおける変数，リテラル，式や問題文中の説明語句とし，これらを ””で囲み，文字列として扱う．

1 つのパラメータ変数で 1 種類の値だけを表すとすると，ばらばらに記述された複数のパラメータ変数同士の関連を把握するために手間がかかる．その手間を減らすためには，問題文中の説明とプログラムの変数名，for 文の初期化式と条件式と更新式など関連するパラメータをひとまとめにして記述すると有効である．関連するパラメータをひとまとめにして扱うために，複数のデータを「組」として表現できるようにした．組は丸括弧 () の中にカンマ，で区切ってデータを記述する．例えば，教科のデータは “("国語", "jap"), ("数学", "math")” のように記述できる．問題文部やプログラム部では組を表す変数に対して，“${変数名:数値}” として，組の何番目のデータを用いるかを指定する．

3.2 節の特徴 1 で例に挙げた問題における科目数のような不定個のデータの集合を扱えるようにするために「リスト」を表現できるようにした．リストは波括弧の中にカンマ，で区切ってデータを記述する．リストの各要素は同じ型のデータでなければならない．教科のデータのリストは “${kamoku-list} = {("国語", "jap"), ("英語", "eng"), ("数学", "math")}” のように記述できる．

組を要素とするリストの変数に対して :数値 を指定した場合は，組の指定した順序のデータのリストとし，簡潔に記述できるようにする．例えば，“${kamoku-list:2}” という記述は “{"jap", "eng", "math"}” の各要素を指す．

### 4.3 制御命令の設計

パラメータ変数を用いるだけでは，似たような記述を複数回繰り返す場合や if 文などの問題によく使われる記述を表現しづらい．テンプレートの記述能力を上げるためには制御命令が必要である．制御命令は，リストの各要素に対する処理や，変数の値で条件分岐できるように設計した．

#### 4.3.1 foreach 命令

3.2 節の特徴 1 のように，教科の入力のために科目数分だけ scanf を繰り返すなど，パラメータ変数の個数に応じて似た記述を繰り返す場合があるが，それをすべて手作業で入力すると入力ミスが起こりやすい．ミスを減らすために，パラメータ変数を受け取り，似た記述を繰り返す制御命令を用意する．リストの要素を変数に順に代入して繰り返し処理するために foreach 命令を用いる．“<% foreach ${変数名} in ${リスト名} %>${変数名} を使った記述<% done-foreach %>” のように記述し，foreach 命令で囲んだ記述がリストの要素の個数分だけ繰り返される．C 言語の配列操作のように，リストの長さを指定した数え上げで繰り返しを記述することもできるが，提案するテンプレートでは，リストの要素の繰り返ししかないので，perl などの foreach を参考にして簡潔に記述できるようにした．

例えば，教科の入出力プログラムは図 1 のように記述できる．

```
<% foreach ${kamoku} in ${kamoku-list} %>
printf("${kamoku:1} ? ");
scanf("%d", &${kamoku:2});
<% done-foreach %>
```

図 1 foreach 命令の使用例

#### 4.3.2 if-else if-else 命令

3.2 節の分析により，プログラム内の if 文を用いた問題を作成したい場合が多いことがわかる．if 文を foreach 命令によって図 2 のように記述することもできるが，この場合はテンプレート部分の記述が煩雑になり，パラメータ部分も 1 つの if-else if-else 文を記述するために複数の変数に分かれて，わかりづらくなってしまう． if 文は問題に使う頻度が高いので，制御命令を作った際に多くの手間を減らすことができる．よって，if-elseif-else 命令として記述し，foreach 命令の繰り返しにおいて，出現順に応じて if 文，else if 文として生成することとした (図 3)．条件の変数の値が"OTHERWISE"の場合は else が生成される．対応するパラメータを図 6 に示す．

```
<program>
if (${if-prm:1}) {
  ${if-prm:2}
}
<% foreach ${elseif-prm} in ${elseif-prms} %>
else if (${elseif-prm:1) {
  ${elseif-prm:2}
}
<% done-foreach %>
else {
  ${else-prm}
}
</program>

<parameter>
${if-prm} = ("条件 1", "処理 1");
${elseif-prms} = {("条件 2", "処理 2"), ("条件 3", "処理 3")};
${else-prm} = "処理 4";
</parameter>
```

図 2 if 文の記述例

```
<% foreach ${rule} in ${pass-rules} %>
<% if-else if-else %> (${rule:2}) {
 printf("${rule:1} です\n");
}
<% done-foreach %>
```

図 3 if-else if-else 命令の使用例

#### 4.3.3 fold 命令

リスト内の要素であるパラメータ変数をすべて使用して記号で繋げたい場合がある．例えば，変数を宣言し，科目の合計を求める場合などが該当する．このような式は foreach 命令では図 4 のように複数の文として記述できるが，生成されたプログラムは不自然となる．

そこで，リストの要素を指定した記号で繋げて出力する fold 命令を定義し，“<% fold 記号 ${リスト名} %>” のように記述する．

#### 4.3.4 if 命令

3.2 節の特徴 3 で挙げた問題では，問題に合わせて空白を出力するかしないかを変更する必要がある．このように，ある変数の値によって文の有無を制御したい場合があるので，これを if を用いて “<% if ${変数名} %>${変数名} を使った記述<% done-if %>” のように記述する．

```
書き換え前：
<% foreach ${kamoku} in ${kamoku-list} %>
int ${kamoku:2};
<% done-foreach %>
total = 0;
<% foreach ${kamoku} in ${kamoku-list} %>
total += ${kamoku:2};
<% done-foreach %>

書き換え後：
int jap;
int eng;
int math;
total = 0;
total += jap;
total += eng;
total += math;
```

図 4 foreach 命令のみを用いて記述する場合

条件は 1 が真，0 が偽とする．例えば，科目の合計点の計算では${need-total}の値が 1 の場合は，合計点を計算する変数や文が生成され，0 の場合は生成されない．

```
<% if ${need-total} %>
int total;
<% done-if %>
int <% fold , ${kamoku-list:2} %>;
<% if ${need-total} %>
total = <% fold + ${kamoku-list:2} %>;
<% done-if %>

${need-total}=0 の場合の出力：
int jap,eng,math;

${need-total}=1 の場合の出力：
int total;
int jap,eng,math;
total = jap + eng + math;
```

図 5 if 命令，fold 命令の使用例

## 5 検証と考察

### 5.1 記述能力

本研究では，一般的な手続き型言語に見られるデータ構造であるリスト，レコード型 (構造体) と制御構造である繰り返し，条件分岐を表現可能であるので，プログラミング問題を生成するテンプレートとして充分な記述能力であると考えられる．

### 5.2 手間の軽減

手間の軽減ができているか議論する．テンプレートを作る必要があるので，授業 1 回分程度の演習問題を作成するには既存の問題を手作業で編集するほうが短時間で済む．自動出題システムで学習する場合には大量の問題を用意する必要がある．その場合，1ヶ所にパラメータがまとまっている本研究のテンプレートを用いると短時間で多くの問題が生成できる．

1 つのテンプレートから多くの問題を生成できる例を 2 つ挙げる．for 文を利用してアスタリスクで三角形や四角形などの図形を出力する問題は，複数の教科書に載っている．この問題をテンプレートで記述する場合，for 文で使う 3 つの式を組として簡潔に記述できる．この問題テンプレートからは，与えるパラメータ 9 通りと繰り返しの方法 2 通りによって，18 通りの図形を描画する問題を生成することができる．

成績判定の問題では，図 6 のようにパラメータを記述することで多くの問題を記述することができる．テンプレート内のパラメータと生成される問題は 1 対 1 で対応しているので，パラメータを増やすことにより，問題を増やすことが可能である．

```
<parameter>
kamoku-list = {("国語", "jap"), ("数学", "math"),
               ("英語", "eng"), ("社会", "soc"),
               ("理科", "sci")};
pass-rules = {("A",  "total>=450", "合計点が 450 点以上"),
              ("B",  "total>=400", "合計点が 400 点以上"),
              ("C",  "OTHERWISE", "それ以外")};
</parameter>
<parameter>
kamoku-list =
{("国語", "jap"), ("数学", "math"), ("英語", "eng")};
pass-rules =
{("合格","total>=210 && jap >= 60 && math >= 60 && eng >= 60",
  "合計点が 210 点以上 かつ 各科目が 60 点以上"),
 ("不合格", "OTHERWISE",  "それ以外")};
</parameter>
```

図 6 成績判定プログラムのパラメータ例

作成したテンプレートから生成することのできる問題数が多ければ多いほど，問題作成者が手作業で問題を作成する場合に比べて問題作成に必要な時間と手間が減り，本研究で提案した問題テンプレートが有効であると考えられる．これらの例より，本研究のテンプレートによって多くの問題が生成できる．

## 6 おわりに

本研究では，プログラムの空欄補充問題を生成するために，プログラミングの問題を分析し，問題テンプレートを用いて生成しやすい基準について考察した．また，問題テンプレート内での変数と制御命令について提案し，どの程度の問題をテンプレート化可能であるか検証した．

今後の課題として，提案方法によって生成したプログラムに対して文献 [1] を利用して実際に空欄補充問題を作成すること，および，テンプレートのより可読性の高い書き方の検討が挙げられる．

## 参考文献


[1] 長谷川靖成，大竹諒，田島侑典，“プログラミング学習における意図を考慮した空欄補充問題の自動生成，” 南山大学 情報理工学部 ソフトウェア工学科 2012 年度卒業論文要旨集.

[2] 伊藤恵，美馬義亮，大西昭夫，“Web サービスを利用した Moodle の課題チェック機能の外部拡張とその実践，” 情報処理学会研究報告．教育学習支援情報システム，Vol.2010-CLE-3 No.6，Dec．2010.

[3] 中島秀樹，高橋直久，細川宜秀，“プログラミング学習のための QA サイクル:受講者の習得度に応じた問題自動提示メカニズム，” 電子情報通信学会論文誌，Vol.J88-D-I，No.2，pp439-450，Feb．2005.

[4] Rails ドキュメント，“ビュー (View)，” http://railsdoc.com/view.

[5] 日本オラクルインフォメーションシステムズ株式会社，“iPlanet Web Server, Enterprise Edition サーブレットに関するプログラマーズガイド，”http://docs.oracle.com/cd/E19957-01/816-2142-01/contents.htm.