

修士論文

C言語穴埋め問題を用いた学習システムの 実現

指導教官 横田 一正 教授

岡山県立大学大学院 情報系工学研究科
電子情報通信工学専攻

有安 浩平

平成22年2月8日

Realization of the learning system that uses Fill-in-the-Blank question for C language

Kouhei Ariyasu

Abstract

It is important by the learning of the C Language to repeat a process programming. Teachers make the learner repeatedly answer the exercise, and are aiming at established of knowledge in the class. However, it takes trouble for a teacher to make a lot of exercises.

In this paper, we propose the learning system that uses Fill-in-the-Blank question for C language. It has a function to generate a Fill-in-the-Blank question for C language automatically. And this system enables study that uses a Fill-in-the-Bland question. It proposes the method of setting questions that switches the problem of setting questions by using a relation and various evaluation approaches of the problem. Furthermore, The system perform the marking that considered the difference of how to write program every learner with a syntax tree.

C 言語穴埋め問題を用いた学習システムの実現

目次

1	序論	1
1.1	本研究の背景と目的	1
1.2	本論文の構成	2
2	穴埋め問題を用いた学習システム	3
2.1	穴埋め問題を用いた学習とは	3
2.2	関連研究	3
2.3	関連研究の問題点	6
2.4	システムの要件	6
2.5	アプローチ	7
3	問題の自動生成	8
3.1	自動生成機構	8
3.1.1	生成する問題	8
3.1.2	問題生成の流れ	9
3.2	構文解析	10
3.3	問題作成意図	12
3.4	問題作成ルール	14
3.5	問題生成	15
3.6	問題の蓄積	17
4	出題の自動制御	19
4.1	出題手法	19
4.1.1	関係性を用いた出題	19
4.1.2	形成的評価を用いた出題	19
4.1.3	総括的評価のための出題	20
4.2	出題の流れ	20
5	自動採点	22
5.1	採点上の問題点	22
5.2	採点手法	23
5.2.1	対応テーブルによる変数の比較	23

5.2.2	正規フォーマットへの統一	24
5.2.3	構文木の比較	25
5.2.4	採点の流れ	26
6	実装	27
6.1	問題の自動生成	27
6.2	出題と採点	28
6.3	システムの全体像	30
7	実証実験	31
7.1	実験方法	31
7.2	実験結果	32
7.3	考察	34
8	結論と今後の課題	36
	謝辞	38
	参考文献	39

1 序論

1.1 本研究の背景と目的

大学の情報系学科や情報系企業の研修などでは、プログラミング言語の教育のためにC言語の教育を重視しており、C言語の科目を必修としているところが多い。C言語の学習の初期段階では、概念の理解はもちろん重要であるが、実際にプログラムを作成する過程も重要である。授業内容を知識として定着させるために、プログラムを作成する過程を繰り返すことでプログラムの動作を理解したりプログラムの作成手順に慣れさせている。

しかし、学習者にやみくもにプログラムを作成させても学習の方向性を持たないままプログラムを作成することになるので知識の定着や概念の理解のサポートができるとは限らない。そのような問題を解決するために、学習者に問題を出題し解かせる演習システムが多く開発されている [1][2][3]。これらのシステムでは穴埋め問題を出題し、その穴の部分を考えさせることにより学習を行っている。穴の部分を考えさせながらプログラムを読ませることにより、ある程度内容が決定されたソースを読むことで学習を効果的に進めることができる。だが、これらのシステムでは問題の作成、出題内容の決定など問題作成者にかなりの労力を必要とする。

また、こうした穴埋め問題生成のための労力の軽減を行うための学習システムも研究されている [4][5]。ソースコードをシステムに登録することにより、ソースコードの一部をランダムに穴に置換し問題として提示するシステムなどである。このシステムでは穴にする場所をランダムで決定しているため、学習効果がある出題ができるとは限らないといった問題点がある。

このような問題点を踏まえ、本研究では、問題生成にかかる労力の軽減と学習者に学習を効果的に進めさせることを目標に学習システムの開発を行った。この学習システムでは主に「問題の自動生成」、「自動出題」、「自動採点」の3つの機能を持つ。

問題の自動生成では、問題作成者は問題の元となるソースコードを提示し、問題作成意図（問題作成者がこんな問題を作りたいというイメージ）を、システムに登録してある問題作成意図の中から選択する。そうすると、システムはソースコードを構文解析し、その構文木の部分木を問題意図通りに穴に置換することにより自動的に穴埋め問題を生成する。自動的に問題生成を行うことに

より問題生成にかかる問題作成者の労力を軽減する。

そして、穴埋め問題を効率的な順番で自動的に出題し学習者に解かせ、構文解析を行い構文木の比較などを用いて学習者ごとのプログラミングスタイルの違いに対応した採点を行う。これらの機能により学習者の学習をサポートし、効率的な学習を出来るようにする。

1.2 本論文の構成

2章で関連する穴埋め問題を用いた学習システムについて述べ、それらの問題点と本研究のアプローチを述べる。3章では問題の自動生成について述べ、4章で出題の自動制御、5章で自動採点の詳細について述べる。そして、6章でそれらの実装について述べ、7章でシステムを用いた実証実験について述べる。最後に、8章で結論と今後の課題について述べる。

2 穴埋め問題を用いた学習システム

2.1 穴埋め問題を用いた学習とは

C言語などのプログラミングの学習において、アルゴリズムの組み立て方を学ぶこと、プログラミング言語での表現能力を育成することなどが最終的な目的となる。

授業などでは單元ごとに例題や演習問題を与え、例題のプログラムを読ませたり、プログラムを実際に組ませることによりプログラムの書き方や動き方を学習させており、それを繰り返すことで多くのプログラムに触れることができ、学習者にプログラムの組み立て方や表現力を学ばせている。さらに、似た内容の問題を何度も繰り返し行うことで知識の定着を図っている。

しかし、演習の時間だけではプログラムを作成させる時間や説明をする時間などが必要になり、多くのプログラムに触れさせれるほどの時間が十分に取れるわけではない。よって、予習や復習などの自主学習でプログラムを組む反復練習を行わせ、そこで多くのプログラムに触れさせる機会を与えることでプログラミング能力の育成に役立てる。

自主学習で反復練習をさせる方法としては穴埋め問題を用いたドリル型の問題提示が多く用いられている。反復練習で穴埋め問題を用いる理由は、穴以外の部分を読み取り解答を考えさせるので学習者にプログラムの流れを追わせる力をつけさせるのに有効だと考えられるからである。さらに学習者がプログラムのソースコードをすべて入力する手間が省けるため、問題に解答するまでにかかる時間が短くなり、多くの問題に触れることが出来るようになる。また、プログラミング学習では処理の流れを考えさせながら、プログラムを読ませることも重要なのでこの問題提示法が良いと考える。

2.2 関連研究

穴埋め問題を用いた学習システムとしては萩原らの記入式 Web 試験システム DrillS-L[1][2]がある。DrillS-Lでは問題データベースに登録されグループ分けされた大量の問題を、簡単な出題指定を行うだけでデータベースの中から条件を満たす問題を探し出し、ランダムに出題させる。簡単な出題指定を行うだけでドリル型の学習が行えるため、学習者に関連のある問題を何度も繰り返し解かせることで知識の定着を図っている。DrillS-LではWebブラウザを用いて

複数人の同時アクセスにも対応し、予習や復習で使われている。

また、玉木らの MAX/C[3] も繰り返し穴埋め問題を解かせることで学習を行うシステムである。MAX/C ではアルゴリズムを用いて 1 つの問題から複数のパターンの問題を出題することが可能である。代入操作を学習させる問題などを扱い、代入する値をアルゴリズムを用いて動的に変更している。MAX/C の出題例を図 1 に示す。出題ごとに問題文中の四角の中の数字が変更され、類似の問題を何度も解くことが可能となる。

変数x、yの値が x y であるときに、
次の代入文を実行すると、実行後の変数の
値はどうなりますか？
x = x + y ;
答え: x y

図 1: MAX/C の出題例

内田らの JavaDrill[4] ではソースコードを登録するだけで自動的に穴埋め問題を出題し、採点を行うシステムの開発を行っており、Java の学習に対応している。このシステムで学習を行わせる場合、問題作成者は多くのソースコードをシステムに登録しておく。学習者が学習を行うときシステムにアクセスすると、穴埋め問題が順次出題される。穴埋め問題を出題するとき穴の位置は乱数を用いて、登録されたソースコード中の単語をランダムに穴をあけ 1 問ずつ提示される。そのため、問題作成者はソースコードをシステムに登録するだけで学習者に学習させることが可能になり、問題作成の手間はかなり軽減されている。

Javaドリル(問題)

【問題番号】 1010

【タイトル】 1から10までの和

【設問】

1から10までの和を求めるプログラムである。

空欄を埋めて完成しなさい。

ファイル名: Sum1to10.java

実行結果:

sum = 55

【プログラム】

```
class  {  
    public  void main(String args[]) {  
        int s = ;  
  
        for (int  = 1; i <= 10; i++)  
            s += ;  
  
        System.out.println("sum = "  s);  
    }  
}
```

送信

図 2: JavaDrill の出題例

2.3 関連研究の問題点

2.2 節で述べた関連システムにも問題点がある。

DrillS-L や MAX/C ではデータベース登録やアルゴリズムの記述などを行って問題を作成・登録しているので，新規に問題を作成するときに問題作成者に多くの負担がかかる．また，独自の記述法で問題が記述されているため，システムで扱う問題の記述方法についての知識を持っている人でしか問題を作成することができない．そのため，その問題作成者に多大な労力がかかる．

JavaDrill ではソースコードを登録するだけで穴埋め問題を作成することができるので，問題作成者への負担はかなり軽減されている．だが，穴の位置をランダムで決定しているため出題する問題のレベルを制御しづらい．また，同一のソースコードでも穴の開ける位置が変わるだけで，入出力の問題になったり，変数の問題に変化したりレベルと同時に出題範囲も変わってしまうため，意図した学習効果がある問題が出題されるとは限らないという問題がある．

2.4 システムの要件

2.3 節で述べた問題点を基に，システムの要件を述べる．

要件 1 問題作成者の負担を軽減する

問題作成者は学習の進行状況に合わせて多数の演習問題を作る必要がある．しかし，問題作成の作業は手間がかかり，問題作成者に多くの負担がかかる．そこで，問題作成時にデータベースへの登録や複雑なアルゴリズムの記述などの手間をなるべくかけないで自動で穴埋め問題を作れる必要がある．

要件 2 問題作成者の意図を考慮した問題の生成

問題作成者は学習目標に沿って穴埋め問題を作成する時，学習目標を満たすために必要な項目を決めている．その学習目標に沿った必要項目に対応した問題のイメージを本研究では問題作成意図と呼ぶ．穴埋め問題を作成する場合，その問題作成意図に合わせた内容の問題生成を行うことが必要になる．問題作成意図はいろいろな抽象度の意図があるため，それに対応する必要もある．

要件 3 学習者の理解度に合わせて出題

演習では，繰り返し問題を解くことで学習効果を高めることができる．プログラミングの問題にはさまざまな種類があり，学習者によってはある特定分野の問題が理解できていないことがある．そこで，理解できていない

内容を推定し、理解できるように導く出題法が必要である。

要件 4 解答の多様性を考慮した採点

プログラムは繰り返し構文の使い方など、同じ動作を行うプログラムでも書き方が複数あるものが存在する。そういった、複数考えられる記法についてどれを答えても正解であると判定されないと正解なのに間違いだと評価されてしまうことが起こる。そこで、その複数の記法を同一とみなし採点を行う必要がある。

2.5 アプローチ

2.4 節で述べた要件を踏まえ、本研究でのアプローチを以下に述べる。今回研究での対象者を C 言語学習の初期段階の学習者とする。

アプローチ 1 要件 1 と要件 2 に対処するために、本研究では問題作成者が C 言語のプログラムソースをシステムに入力し、出題したい問題イメージを問題作成意図という形で提示することにより、自動的に穴埋め問題を生成する。

アプローチ 2 要件 3 で述べた問題について、問題 1 問ごとに評価を行い次に
出題する問題の制御を行う。また、問題をグループ分けすることにより間違えた問題から、学習者の理解度を推測し出題レベルの変更や復習させるなどの制御を行う。

アプローチ 3 要件 4 で述べた問題については、構文解析を利用した採点を行う。模範解答のソースと学習者が答えた解答のコードを両方とも構文木に変換し比較する。そうすることにより、構造の差異を取得することが可能である。また、構文エラーなどの判定も構文解析時に出来るので迅速な採点が可能である。また、違う記法で同じ動作を行うものに対しては統一のフォーマット（正規フォーマット）に統一することで、その差異を吸収する。

3 問題の自動生成

この章では2.5節で述べたアプローチの穴埋め問題の自動生成について述べる。問題の自動生成は穴埋め問題自動生成システム [6][7] の問題の自動生成部分をもとにして、より簡単に問題作成を行えるよう機能追加を行い作成している。

3.1 自動生成機構

3.1.1 生成する問題

プログラミングの演習問題ではプログラムを一から記述させる記述問題、プログラム中のあるまとまりを空欄にし、空欄により隠された部分を推定し記述させる穴埋め問題、穴埋め問題の答えを複数の候補の中から選択する選択問題など複数のバリエーションがある。

しかし、記述問題ではプログラムを一から作らせるため、学習初期の学習者にはハードルが高く学習意欲をそぐ可能性がある。さらに、プログラムを問題文のみから設計して作らせるため、プログラム設計が上手くできず問題作成者の意図した学習効果が得られない可能性がある。また、選択問題では複数の選択肢の中から答えを選ぶので適当に答えても正解になる可能性があり、ここでも意図した学習効果が得られないことがあると考えられる。

そのため、今回はプログラムの一部分を穴にしてその部分のコードを書かせる穴埋め問題が有効であると考え、穴埋め問題ではプログラムのソースを読み穴の部分を考えるため、学習者の実力を判定するだけでなく、ソースコードを読む力が養われたり、プログラムの書き方の例を示すことが出来る。その上、ある程度のプログラムを学習者に示すため、的外れなプログラムを書くことはなく、選択問題のように選択肢がないために勘で答えるといったことが難しなる。さらに、穴を大きくすることにより難易度の高い問題にすることもできるため、扱える問題の難易度も穴の大きさなどにより様々に設定することが可能である。

本論文では比較的小さな穴の穴埋め問題を用いて学習を行うこととする。生成する問題例を図3に示す。ソースコード中の [A] が穴となっている場所である。

問題文

[A]に当てはまるコードを書きなさい。

ソースコード

```
#include<stdio.h>
int main (){
    int a;
    scanf("%d",[ A ]);
    printf("%d\n",a);
    return 0;
}
```

[A] =

図 3: 生成する問題例

3.1.2 問題生成の流れ

問題作成では図 4 で示す流れで自動的に穴埋め問題の生成を行う。まず，問題作成者は問題にするためのソースコードを用意する。用意したソースコードをシステムに読み込ませると構文解析が行われ構文木に変換される。続いて問題作成者は後述する問題作成意図を選択し，どんな問題を出題したいのかを決定する。システムでは問題作成意図を対応した問題作成ルールに変換する。そして，構文木と問題作成ルールにより，自動的に穴埋め問題が作成される。その後，作成された問題は構文解析で生成した構文木と一緒に問題データベースに登録される。

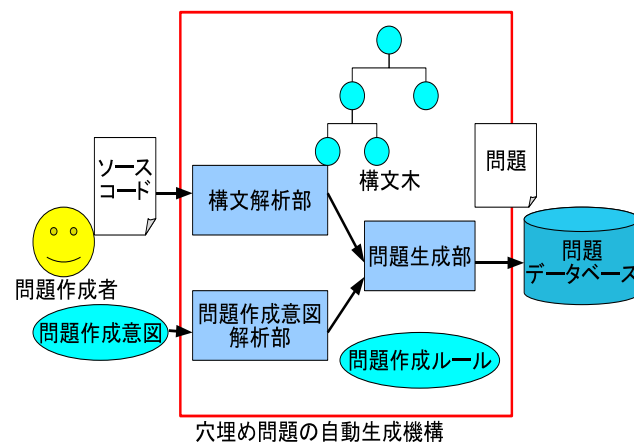


図 4: 問題生成の流れ

3.2 構文解析



C 言語のソースコードをシステムの入力とする。入力したソースコードは、構文解析が行われ構文木の形に変換され、XML 文書で記述する。XML 文書化することで以下に述べる変数 ID などの情報を持たせることが出来るようにする。また、同時に変数の対応テーブルが作成され、すべての変数には変数 ID を付加する。さらに、その対応テーブルに準じた形で変数を表す葉には属性として変数 ID を付加する。

構文解析によって得られる構文木の主な仕様を述べる。

根 構文木の根はプログラム 1 つにつき 1 つのみ存在し、値は null を持つ。直下の子には唯一 SOURCE 節のみ持ち、他の要素が根の子になることはない。

節 C 言語プログラムの構造を分かりやすく表すために付加される。たとえば、ソースコード全体を表す SOURCE 節、宣言部分を表す DECLARATION 節、main 関数などの関数を表す FUNC 節などがあり、その節の子にそれらが示す内容を表している。また、プログラムの一部分も節になり、代入演算子や論理演算子、構文なども節に置換される。この節は主に意味のある単位で部分木を抽出するために使用される。問題生成では問題作成ルールで指定された部分木を穴に変換しているので、その部分木の指定時に節をもとに探索される。

葉 構文木の葉となるのは、定数、変数名、型名、関数名、以下に部分木を持たないことを示す NONE などになる。

節はプログラムの構造を示すラベル節とソースコードの一部分を置換する置換節に分かれる。主なラベル節を表 1 に、主な置換節を表 2 に示す。

表 1: 構文解析木の節（ラベル）

節の名前	節が表す意味
DECLARATION	宣言
DECLARATOR	宣言子
TYPDEF	型宣言
TYPE	型名
FUNC	関数
GLOBAL	グローバル領域
BLOCK	{ } で囲まれた部分
STATEMENT	文
EXPRESSION	式
FORMAT	フォーマット指定子
ARGUMENTS	引数群
ASSIGNMENT	代入
LASSIGNMENT	代入の左辺
RASSIGNMENT	代入の右辺
RVALUE	右辺値
ARRAY	配列
STRUCT	構造体
MEMBER	構造体のメンバ
NONE	式の省略

表 2: 構文解析木の節（置換）

節の名前	節が表す意味
FOR,DO,IF...	各構文
ASSIGNMENT	代入
PLUS	加算
MINUS	減算
UNARYPLUS	前置インクリメント
UNARYMINUS	前置デクリメント
POSTFIXPLUS	後置インクリメント
POSTFIXMINUS	後置デクリメント
AMP	アドレス演算
DOT	ドット演算
ARROW	アロー演算子
EQUAL	等価演算子
FEWER	比較演算子
AND	論理和
NOT	否定

構文解析を行った例を図 5 に示す。この図では上のソースコードを構文解析した結果の構文木が右上に示したものになり、同時に作成された変換テーブルが左下に示すものになる。

```
#include<stdio.h>

int main(){

    char halo[] = "Hello!";
    printf("%s\n",halo);

    return 0;
}
```

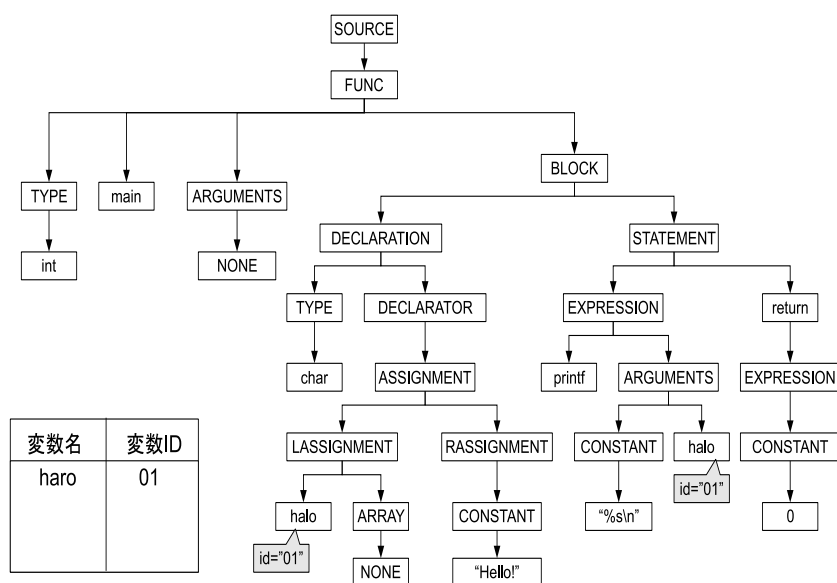


図 5: プログラムの解析例

3.3 問題作成意図

問題作成意図とは、問題作成者が「こんな目的で問題を作りたい」と考えている内容であり、ある学習目標に対し、「正解できれば、これが理解できている」という指針になるものである。この問題作成意図は、満たしたい学習目標ごとに様々なものがある。

例えば、構文について学習させたい時、問題作成意図としては「出力文が分かっているのか確認したい」、「for 文の使い方が分かっているのか確認したい」、「処理の流れが分かっているのか確認したい」等さまざまな問題作成意図が挙げられる。それらの意図を選択することにより、意図に対応した問題が作成されるようにする。

今回はプログラミングの学習初期の内容についての学習意図について考えた。表3は問題作成意図の例である。

表 3: 問題作成意図の例

入力関数の引数が分かっているか
制御構文の引数が分かっているか
出力関数の引数が分かっているか
変数がうまく定義できているか
関数を使うのに必要なヘッダが分かっているか
必要となる引数が分かっているのか
所定の演算子を使えるか
処理の流れを理解しているか
配列の定義のしかたが分かっているのか
フォーマット指定子が分かっているか
演算子の使い方が分かっているか
どこでどの変数を用いなければならないか分かっているか
返り値が分かっているか
必要な関数が分かっているか
処理内容が分かっているか
自分でプログラムを作れるか
出力結果のイメージができるか
出力関数の動きが分かっているか

表3で表された意図は抽象的で、具体的にどんな問題を作成するのか決まっておらず、問題作成意図から問題を作成するのは難しい。そこで、この問題作成意図は、~~単純化・具体化するために、以降に説明する問題作成ルールを組み合わせることによって表現す~~問題作成意図と問題作成ルールの対応付けはシステム側で用意する。しかし、それでは作りたい問題ができないと問題作成者が思った場合、問題作成者が問題作成ルールを組み合わせることで新規に問題作成意図を作成することも可能である。

3.4 問題作成ルール

問題作成ルールは構文木のどの部分を穴にするかを指定する記述である。このルールに従って問題生成部で構文木の部分木が穴に置換され問題が作成される。問題作成ルールは以下のような記述を持つものになる。この記述を複数個組み合わせることにより問題作成意図を表現する。

createblank(節名, 指定 (, 指定内容))

節名には表 1, 表 2 で示した構文木の節が来る。この指定した節以下を穴埋めの穴と置き換える。

指定には”all”または”select”が入り, ”all”の場合は節名で指定した節の下はすべて穴になる。また, ”select”を入れた場合は, 後の指定内容で指定したとおりの穴のあき方になる。

指定内容では, 節名で指定した部分の中でも一部分にだけ穴を開けたい場合に指定する時用いる。しかし, この部分指定では構文解析で分割された単位以下の指定はできない。ここで用いることが出来るものは, rootonly(節名となっている部分のみを書き換える), nameonly(子要素の変数名のみ書き換える), numberonly(変数の値のみ書き換える), conditiononly(条件式のみ書き換える)などがある。指定内容の例を表 4 に示す。

表 4: 指定内容例

指定内容	実行内容
rootonly	節名となっている部分を穴にする
leftchild	指定節の左の子を穴にする
rightchild	指定節の右の子を穴にする
nameonly	子要素の変数名を穴にする
numberonly	変数の値を穴にする
conditiononly	条件文を穴にする
statementonly	構文の実行式を穴にする

3.5 生成

問題は3.2節で述べた構文解析の結果の構文木と3.4節で述べた問題作成ルールを用いて問題を生成する。問題生成の流れを図6に示す。

最初に、選ばれた問題作成ルールから構文木のどの部分を穴に置き換えるかを探し出す。穴に置き換える部分を探すのには、まず問題作成ルールの「節名」を読み、幅優先探索を行い構文木から「節名」と一致する節を見つける。次に「指定」のところに記述された内容に従って、"all"の場合は一致した節を根とする部分木すべてを穴の記述に置き換える。"select"の場合は「指定内容」の部分で指定された実行内容に従い一致した節を根とする部分木を穴の記述に置き換える。それをすべての問題作成ルールに対して順に適用させていき、穴の埋め込まれた構文木を作成する。

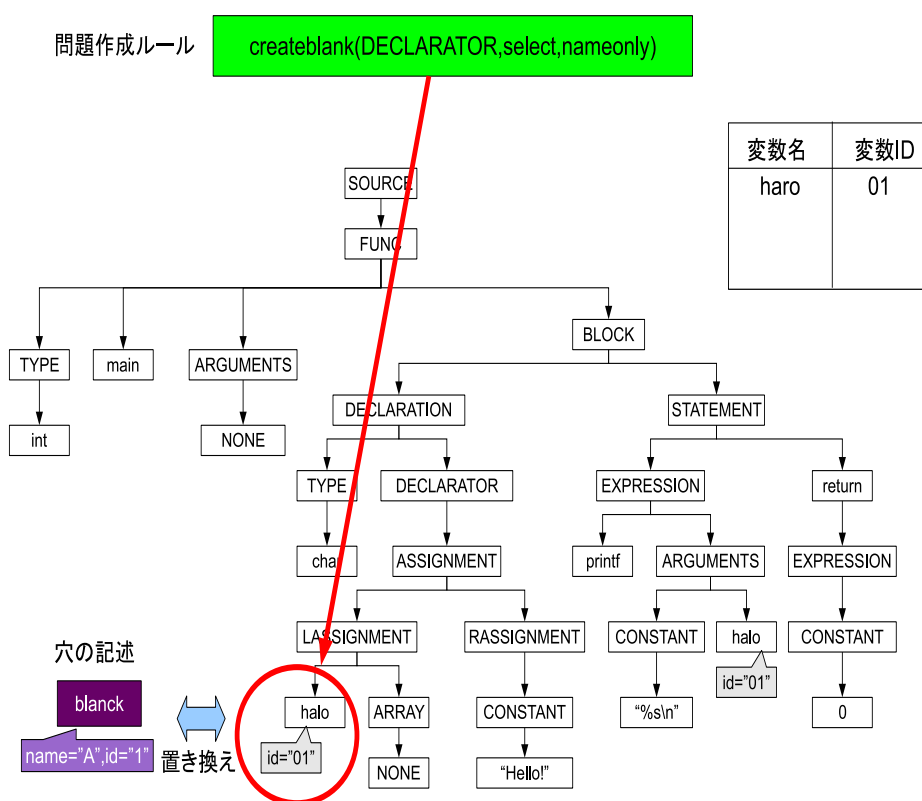


図 6: 問題生成の流れ

そして、出題には ADEL のテスト機構 [8] での問題の流用を想定して、構文木をテスト機構でも扱えるテスト記述に整形する。このテスト記述の構造は後に示す XML の形式になっている。生成する問題には問題ごとに問題 id をふり、グループ分けしたうえで、後の 3.6 節で述べる問題データベースに格納する。

テスト記述の内容について以下に示す。

program_set 問題のグループを持つ要素で、問題グループを識別する id 属性を持つ。子要素には複数の item 要素を含む。

item 問題候補のルート要素であり、固有の識別子である id 属性と問題の形式を表す type 属性をもつ。item1 つか問題 1 問に相当する。子要素に question 要素、response 要素、hints 要素、explanation 要素、evaluate 要素をもつ。

question 要素 問題の内容を記述する要素で、穴が開けられたソースコードはここに記述する。

response 要素 提示の際に解答の候補として記述され、穴埋め問題の解答欄の数に比例する。

hints 要素 問題に対するヒントが記述されている。

evaluate 要素 採点に関する事柄を記述する要素。子要素に function, correct, score, weight, point を持つ。

correct 要素 正解となる文字列を記述する。この要素と解答を比較し正解判定を行う。

point 要素 この問題を正解すると加算される点数を記述する。

explanation 要素 採点後に表示される文字列を記述する。

テスト記述の例を図 7 に示す。

```

<program_set id="print_func">
  <item id="1" type="fill_in_the_blank">
    <question>
      <p>[ A ]にあてはまる語句を入力しなさい</p>
      <p>#include<stdio.h><br/>
        <br/>
        int main(){<br/>
          char halo[] = "Hello!";<br/>
          printf("%c\n",[ A ]);<br/>
          return 0;<br/>
        }</question>
      <response id="1">[ A ]=</response>
      <hints>配列を表示するにはどうするか思い出して！！</hints>
      <evaluate>
        <function>60</function>
        <correct id="1">halo</correct>
        <score>1</score>
        <weight correct="1" incorrect="-1" />
        <point>10</point>
      </evaluate>
      <explanation>lhaloは文字型配列の先頭アドレスを指しています。</explanation>
    </item>
  </program_set>

```

図 7: テスト記述の例

3.6 問題の蓄積

問題の自動生成により作られた問題を出題機構で用いるためデータベースへ蓄積を行う。問題を蓄積するデータベースを問題データベースと名付けた。

問題の自動生成機構で作成された問題は 3.5 節で述べられた XML の形でデータベースに格納する。データベースに蓄積するとき、問題には問題の識別子の問題 ID が付加する。問題 ID は問題を読み出す時に用いる識別子となり、複数の問題で重複することはない。

また、プログラミング言語は蓄積型の分野で以前に学習した内容とその時学習している内容に関係がある。その関係性をもとに出題を行うため、問題を内容ごとにグループ分けを行い、プログラムセットを作成している。ここでの



プログラムセットとは、同様な問題内容の穴埋め問題の集合を指し、どのプログラムセットに属するかという情報をデータベースに登録時に付加する。属するプログラムセットの決定はプログラムの内容に従って教材作成者が決定する。さらに、同一のプログラムセット内で穴埋め問題の難易度により問題作成者がレベルを決定し付加する。今回は単純化のため3段階のレベル設定とし、問題を受ける学習者の8割以上解ける問題をレベル1、5割程度が解ける問題をレベル2、3割程度の学習者しか解けない問題をレベル3と想定する。

問題と同様に、3.2節で作成された構文木も、生成時に作られた構文木のXMLの形式で後に述べる正規フォーマットに変形を行い、模範解答としてデータベースに登録する。

4 出題の自動制御

4.1 出題手法

本章では3章で述べた自動生成により生成した問題の出題手法について述べる。出題は学習効果を高めるために問題の関係性に着目した「関係性を用いた出題」、教育的な評価手法の形成的評価と総括的評価を用いた「形成的評価を用いた出題」、「総括的評価のための出題」の3つの出題法を組み合わせた出題を行う。

4.1.1 関係性を用いた出題

工学分野の教材は知識を蓄積していくことで学習が進行していく。C言語でも同様に文法やアルゴリズムなどの知識を蓄積していくことで作ることができるプログラムが増え、様々な要求に対応できるプログラムを書くことができるようになっていく。


このような蓄積型の分野の教材では前に学習した事柄との関連性が強く教材ごとのつながりが強いものが多い。たとえば、forなどの繰り返し構文を習う場合、事前に変数についての知識がないと上手く使いこなすことが出来ない。また、同じ学習内容の中でも段階的に知識をつけていくことがあり、順番に解いていくことで理解しやすくなる。ここでの例はポインタの内容での「ポインタ」と「ポインタ配列」などである。ポインタについて先に習うことで、ポインタ配列を学習するときに以前の知識をもとに学習できるので理解が早くなる。

よって、このような関係性をもとに問題を内容ごとにグループ分けを行い、そのグループごとに問題を行う。このグループ分けを用いた出題は3.6節で述べたプログラムセットを用いて行う。プログラムセットは問題の登録時に同様な内容の問題ごとにグループ分けされており、同一のプログラムセットであれば関連した問題が出題される。

さらに、問題登録時に同一のプログラムセット内で問題の難易度ごとにレベルが付加されている。このレベルを用いてやさしい問題から順に解くことにより理解がしやすくなる。

4.1.2 形成的評価を用いた出題

形成的評価とは小さなテストなどを繰り返し、学習者の学力の移り変わりを評価する評価手法である。学習の途中に小テストを行いどの程度理解できているのか評価を行う。短い間隔でテストを行うことで細かく学習者の理解度に対

応しながら学習を進めていくことが出来、学習進度と学習者の理解度がマッチしないとといった問題を防ぎ、学習者の理解度に合わせた学習を行うことが出来るようになる。を出題に応用することにより、学習者の理解度の変化に合わせた出題制御が行う。

本研究では、この形成的評価を出題に組み込むために、1問解くごとに正誤判定をし、その場で次に出す問題のレベルの切り替えを行うようにした。1問ごとに判定を行うことにより、学習者の理解度に応じた問題が出題されるようになり、適切な問題が出題される。このレベルの切り替えでは、上（レベルアップ）下（レベルダウン）2方向の切り替えとそのまま同じレベルで学習を行う3通りの制御を行うようにし、最終的に自分の理解度に近い問題が出るように集約されるように設計した。


さらに、低いレベルで連続して間違えるような学習者には、基礎知識の学習が足りないと判断し、規定数の問題を解いていなくても強制的に復習をさせるようにする。

4.1.3 総括的評価のための出題

総括的評価では単元の最後に学習した内容についてまとめてテストを行い、学習者の学力を評価する評価手法である。最後に総まとめのテストを行うことにより、これまでの学習でのトータルの学習効果の評価を行う。よって、最終的な学習目標に対して学習者がどの程度まで理解出来ているのかを計るのに用いることができ、学習者の学習の全体でみた進捗度合いを把握することが出来る。これを出題に応用することによりある程度の学習目標にまで到達しなかった学習者に再学習を行わせるなどの制御が行えるようになる。

本研究では、教材作成者が指定した問題数と閾値をもとに、指定した問題数を解き終わったときに正解数と閾値を比較して次に行う処理を決定する。この時、正解数が閾値以上だとその単元を終了させ、閾値以下だった場合復習を行わせるように制御し、再学習を行った後もう一度問題に解答させるといった処理をさせることが出来るようにする。

4.2 出題の流れ

本研究はこれまで述べてきた出題手法を組み合わせた独自の出題手法を用いている。出題の流れは図8で示されるような形で出題を行う。

学習者はまず出題される問題の分野の選択を行い、グループ分けされた問題

の中から学習したい分野を決定する。まず、1問目の問題は分野の中で最低レベルの問題群の中からランダムで出題する。ここで、**始め**最低レベルの問題群の中から出題する理由は、簡単な問題から順に段階的に解いていけるため、基礎知識を確認したうえで応用問題を解かせることが出来るからである。続いて、出題された問題に対して学習者が解答を行うと、その場で正誤判定を行う。その正誤判定の結果をもとに形成的評価を行い、次に出題する問題のレベルの制御をし出題を行う。形成的評価では学習を始めた時点からこれまでの正解数と不正解数が事前に問題作成者が指定した閾値と比較を行い、閾値以上になると、次に出題する問題のレベルの変更を行う。正解数が正解数の閾値を上回った場合は問題のレベルを上げ、不正解数が不正解数の閾値を上回った場合はレベルを下げる。正解数、不正解数ともに閾値を下回った場合は、次の問題も同じレベルで出題を行う。また、連続で間違いを続け、事前に設定された連続間違い数の閾値を超えると出題数が残っている場合でも強制的に復習を行わせることも可能にする。

そして、指定した問題数を解くと総括的評価が行われる。総括的評価では正解率の計算を行い、その値が閾値を超えていた場合、単元の終了とする。閾値を超えていない場合は復習を行わせるため再学習を行わせ、もう一度問題を解かせる。

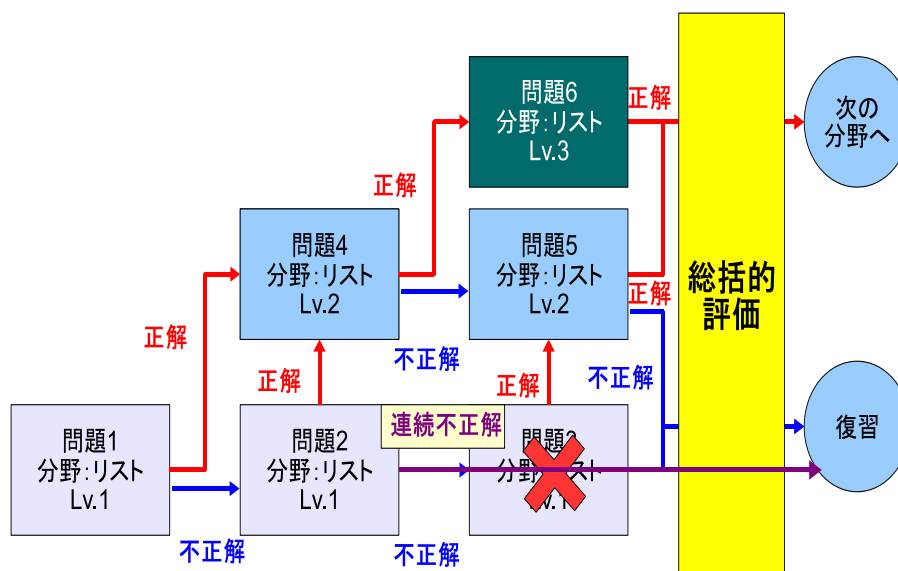


図 8: 出題の流れ

5 自動採点

5.1 採点上の問題点

穴埋め問題の穴^③を大きくした場合、プログラムによっては複数の解答が考えられるものがある。これは、C言語では違う記法で同じ動作を書くことができるために、学習者のプログラムの書き方や知識の違いから起こる。異なる記法で同じ動作を行うプログラムを用いた問題の解答に対してADELのテスト機構[8]では、問題作成時に問題作成者が複数の模範解答を登録することで対応してきた。さらに、自動生成システム[6]においても複数パターンの解答がある場合には問題作成者が手動で登録するという手法をとっていた。しかし、この手法では解答のパターンが多く考えうる場合、すべてのパターンに対して解答を登録してもらうことになり、問題作成者に多くの負担がかかっていた。また、問題作成者任せで複数パターンの解答に対応していると問題作成者が登録し忘れた場合、学習者が答えた解答自体は正解なのに不正解になるといったことが起こりうる。これでは、プログラミングスタイル^③より正誤判定時に不平等が生じる危険性が考えられる。違う記法で同じ動作のプログラムの例を図9と図10に示す。

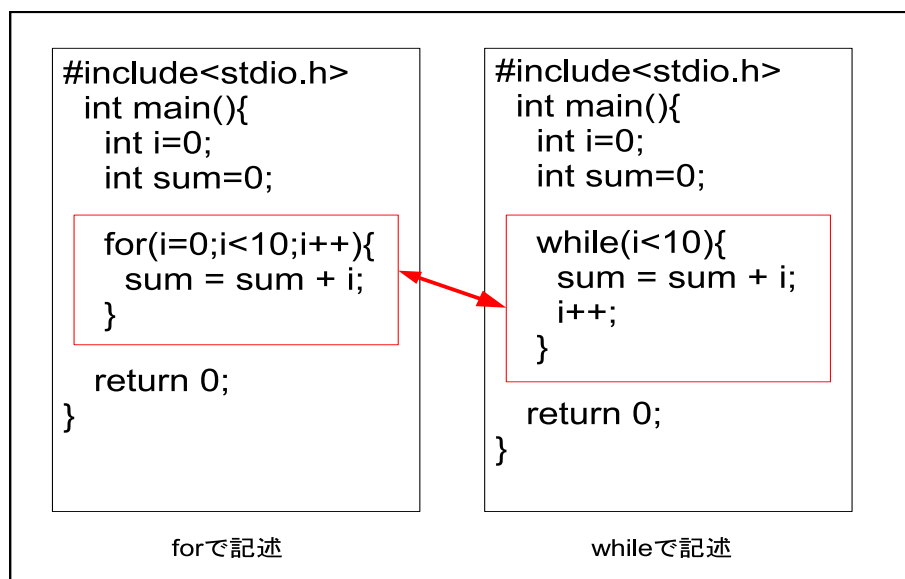


図9: 違う記法で同じ動作のプログラム例 (for と while)

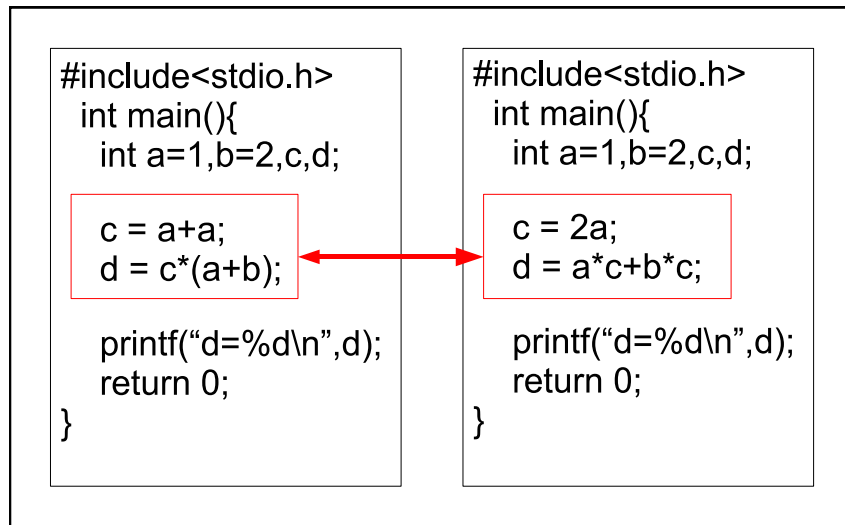


図 10: 違う記法で同じ動作のプログラム例（結合律・分配律）

5.2 採点手法

そこで、5.1 節で述べた問題点について対応した採点手法について述べる。採点機構では、「対応テーブルによる変数の比較」、「正規フォーマットへの変換」、「構文木同士の比較」といった 3 段階の過程で採点を行う。

5.2.1 対応テーブルによる変数の比較

プログラムが目的通りの動作をするためにはこの場所にこの変数が入らないといけないといった制約がある。その変数ごとの制約をまとめたものをここでは制約関係と呼ぶ。変数の採点において、この制約関係は重要な要素になりソースコード中の同一変数をすべて穴にした場合、模範解答と変数名が異なっているとしても、制約関係が合っていれば正解にしてもプログラムとしては問題がない。だが、ADEL のテスト機構の採点などではそういった問題の場合、制約関係があっても模範解答と変数名まで同じでないと正解にならないので問題となっていた。よって、変数名が模範解答と異なっているとしても制約関係が合っていれば正解にする必要がある。

そこで、変数の制約関係に基づき正誤判定を行う。まず、構文解析を行うときに変数に id をつけ、変数の対応テーブルを作成しておく。その対応テーブルを用いて変数の位置の制約関係が共通か判断を行う。構文木の変数のノード ID を比較し、関係を判定する。

5.2.2 正規フォーマットへの統一

5.1 節で述べたように、異なる記法で同一の処理を表すものがある。異なる記法で同一の処理を表す記述に対して問題作成者にすべてのパターンの模範解答を用意してもらうのは手間がかかる。そこで、違う記法で同じ動作を行うコードに対して、ある統一された記法に変換を行う。本研究では、その統一された記法を正規フォーマットと呼ぶ。違う記法で同じ動作を行うものは様々な種類がある。たとえば、 $c(a+b)$ と $ac+bc$ などの分配律は異なる記法で同一の動作を表す。また、 $a+b$ と $b+a$ のような可換律も同様である。そこで、今回繰り返し構文、分配律、結合律、可換律について正規フォーマットに変換を行う。この正規フォーマットへの統一は構文解析をした構文木に対して行う。

繰り返し構文は for 文と while 文について対応する。これらはそれぞれ同じ動作を記述可能なので書き換えることが可能である。この構文については for 文に統一する。

分配律は先ほど述べたようなカッコをつけた形でも、外した形でも正解となるようにする。分配律では正規フォーマットに統一するとき、カッコを外した状態「 $c(a+b)$ の場合だと $ac+bc$ 」に統一する。

結合律では括弧を用いて計算の順番を変えても結果が変わらない、 $a(b*c)$ と $(a*b)c$ などの加算や乗算を行うときのカッコの位置の記述の違いなどに対応する。さらに、 $a+a$ と $2a$ などの記述についても対応する。前半の括弧の位置については括弧を外した状態に統一する。先程の例だと $a*b*c$ という形に変換する。また、後半の $a+a$ と $2a$ などの記述については $2a$ の形に統一する。

最後に、可換律は位置を交換しても計算結果が同じものを表し、正規フォーマットに統一するとき、計算式内の交換できる変数を変数名の辞書順に並べ替える。また、プログラム中の文同士が位置を交換しても計算結果が同じような場合には、後に述べる構文木の比較により対応する。

正規フォーマットへの変換は、部分木の置き換えによって行う。変換の流れを図 11 に示す。構文解析により生成された構文木を用いて、変換パターンの部分木と比較しパターンに一致する場所を見つけだし、置き換え後の部分木のひな形に変数名などをあてはめ、置き換えることで正規フォーマットへの変換を行う。変換パターンの部分木とは、これまで述べてきた変換対象となる構文を部分木の形にしたものである。変換対象となる構文木の変数名や for 文などの条件式以外の部分木の構造と各節名が、この部分木と一致すれば一致する部分

が変換対象となる。変換するパターンの部分木と変換後の部分木のひな形は事前にシステムが用意している。

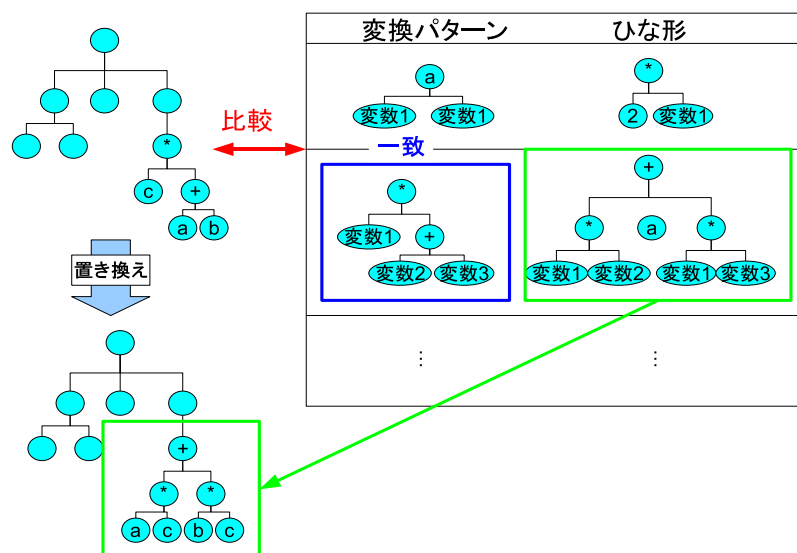


図 11: 正規フォーマットへの変換の流れ

5.2.3 構文木の比較

構文木の比較処理では、正解判定を行う学習者の解答の構文木と問題作成者が作ったソースから生成した模範解答の構文木の2つを用いて構文木の比較を行う。そうすることにより、構造的な差異を見つけることが可能となる。また、プログラム中の文同士が位置を交換しても結果が同じような場合にも対応することが出来る。これら2つの構文木は正規フォーマットに統一された形で比較される。

比較方法は幅優先に節の比較を行い、根から順に対応する節を決定していく。対応する節の決定のために行う比較では、節名で文字列比較を行い、一致すれば対応する節とする。また、一致する節が複数ある場合は一番初めに一致した節と対応させるようにする。一度一致した節は次の節比較には含まないように除外し、1つの節が複数の節と対応するという事態を防ぐ。

この節の比較を解答の構文木のすべての節に対して行い、構文木ごとの節の対応関係を作成する。その対応関係において、対応しない節があればそこを間違いとして学習者に提示する。

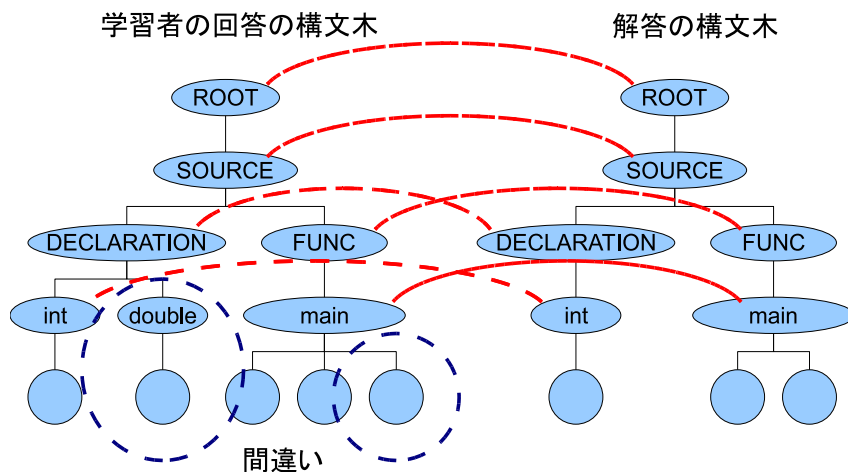


図 12: 構文木比較の例

5.2.4 採点の流れ

学習者が採点ボタンを押したときに採点が始まる。まず、穴を埋めた解答と問題文のソースコードをもとに問題文の穴の部分に解答を埋め、解答のソースコードを作成する。作成したソースを構文解析し構文木の作成を行う。この時、構文解析に通らなかった場合プログラムの内容に間違いが含まれているので、学習者の解答は間違いだと判断する。続いて、学習者が答えた解答の構文木を正規フォーマットに変換する。そして、問題データベースから模範解答の構文木を取り出し、学習者の解答の構文木と比較を行い、構造の違いのチェックを行う。また、変数の位置をすべて穴をあけた場合は対応テーブルをもとに、学習者が入力した文字が問題作成者の意図した文字列でなくても、必要な場所にすべて同じ文字列が入っているかどうか判定する。

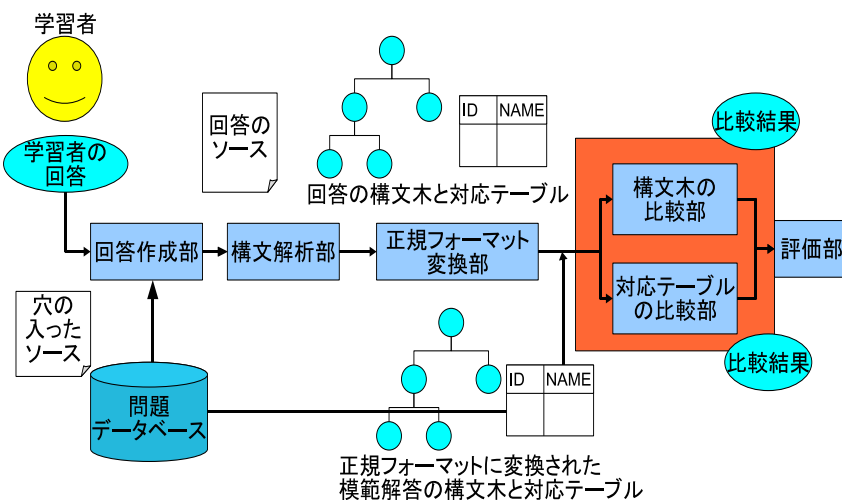


図 13: 採点の流れ

6 実装

本章では実装した穴埋め問題を用いた学習システムのプロトタイプについて述べる。

6.1 問題の自動生成

問題の自動生成機能について、まず構文解析は ANTLR[9] を用いて作成した。ANTLR を用いた理由は開発を行うための優秀なベンチマークがあり、デバッグ環境も充実していたからである。開発言語には様々な OS 上での利用を考え Java[10] を用いた。問題作成ルールや問題作成意図の解析機構、問題生成機構は Java を用いて実装を行った。また、問題データベースには PostgreSQL[11] を用いた。

続いて、問題の自動生成機能の動作例を示す。まず、図 14 に示すソースコード選択画面から問題作成に用いるソースコードを選択^①。続いて図 15 の問題作成意図選択画面でどんな問題を作りたいか選択する。^②ここで複数の問題意図を選択することもでき、問題作成意図ごとに問題候補が作成される。その後、図 16 の画面が提示行われ問題候補の中から実際にデータベースに登録する問題を選択し、問題文や制限時間などの情報を登録する。情報の登録は図 17 の画面で行う。問題に情報を登録後に送信ボタンを押すとデータベースに選択された問題の情報が登録される。

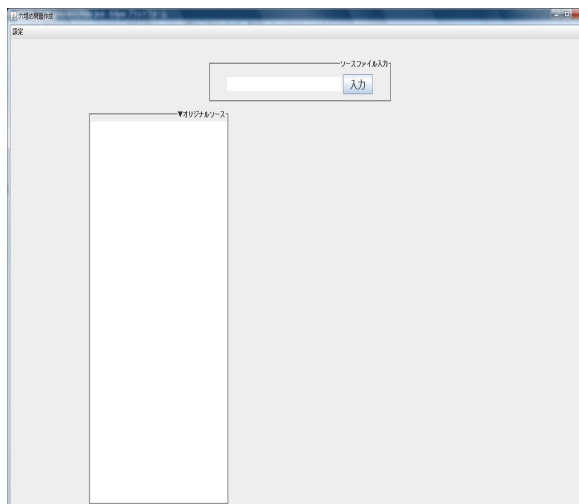


図 14: ソースコード選択画面

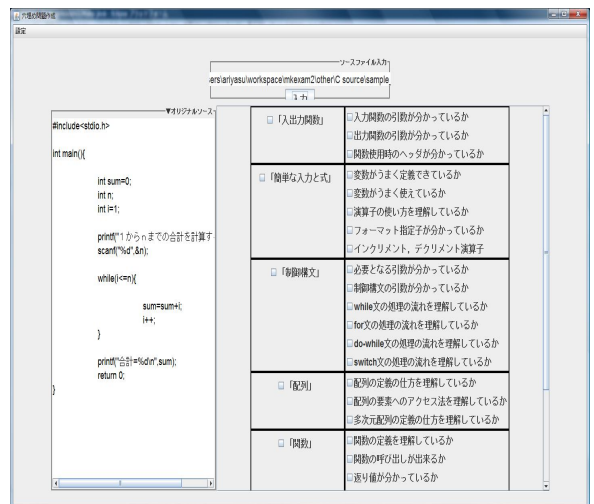


図 15: 問題作成意図選択画面

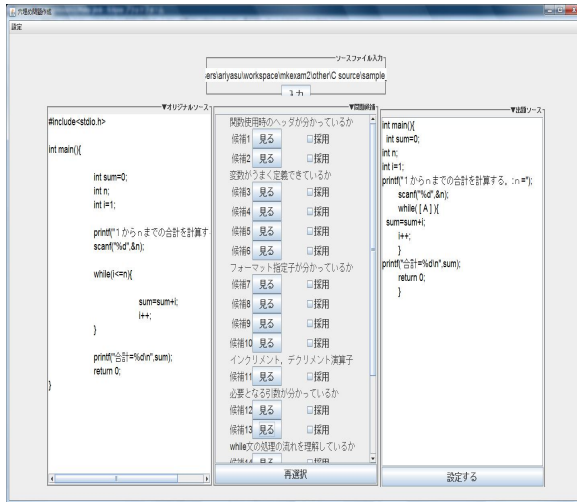


図 16: 問題候補選択画面

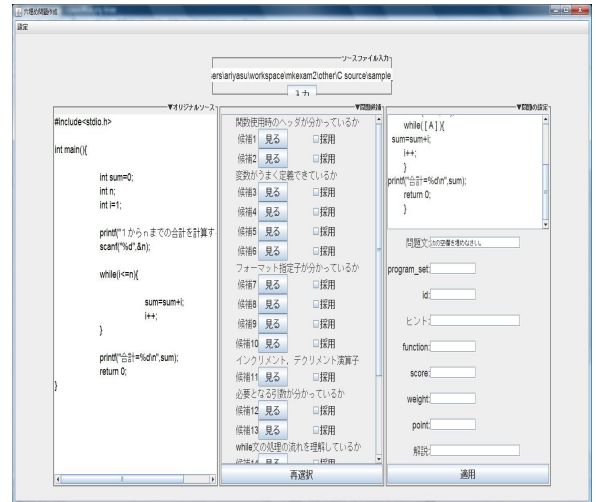



図 17: 問題情報登録画面

6.2 出題と採点

出題・採点機能は Java[10] を用いて実装を行った。学習者の PC 上で動作するプログラムで、問題は JDBC[12] を用いてサーバ上にある問題データベースより問題を取得する。また、同様に学習ログをデータベースに書き込む機能も持つ。

続いて、6.1 節同様に 出題・採点機能の動作説明を  まず図 18 に示す画面から、学習を行う内容を選択し、学習を開始する。ここで選択された内容に従って問題が出題され、この学習内容は事前に教材作成者によって決定されているとする。問題の出題は図 19 のような形で出題が行われる。左側に穴入りの問題文が提示され右側に解答欄と採点ボタンが示される。さらに問題ごとに制限時間を設けており、指定時間以上経過すると強制的に解答が終了するようにした。

採点ボタンを押すと採点が行われ図 20 の画面が提示される。問題同様左側に問題文が表示され、画面の右側には学習者が入力した解答と模範解答が表示され、間違えたときに答えとの比較が出来るようにしている。また、教材作成者からの解説も表示できるようになっている。そして、この画面上の次へボタンを押すと次の問題が表示されるようになっており、出題画面と採点画面を交互に繰り返す形で学習が進むようになっている。

最後に、問題作成者が指定した問題数解くと図 21 で示す画面が提示され、単元の終了になる。終了画面では、その単元での正解数や正解率が確認できるようになっている。

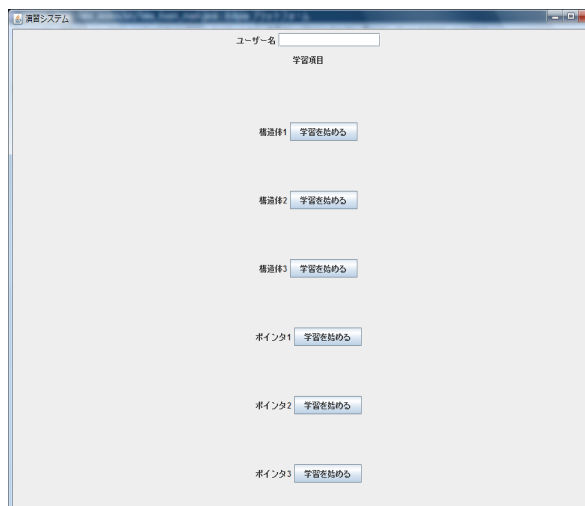


図 18: 学習コース選択画面

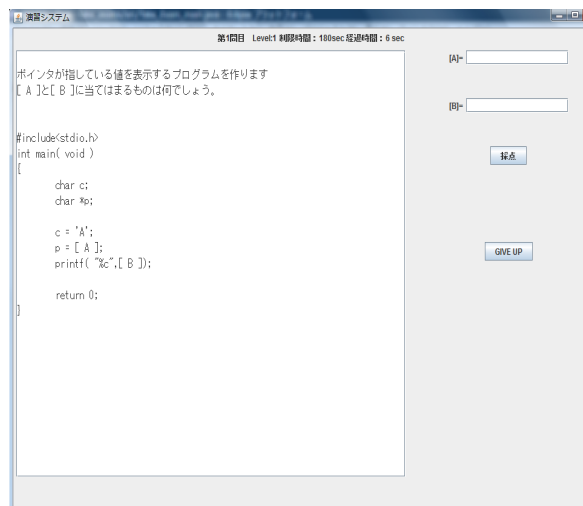


図 19: 出題画面

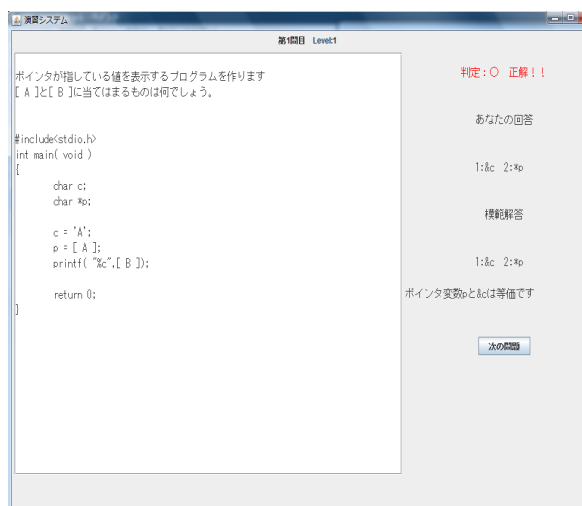


図 20: 採点画面



図 21: 終了画面画面

6.3 システムの全体像

本研究で実現した穴埋め問題を用いた学習システムの全体像を図 22 に示す。提案システムは大きく分けて穴埋め問題の自動生成部と出題・採点部の 2 つに大きく分かれている。穴埋め問題の自動生成部は構文解析を行う部分と問題ルールを決定する部分、問題生成する部分に分かれている。出題・採点部は出題制御を行い問題を出題する部分と採点を行う部分に分かれている。

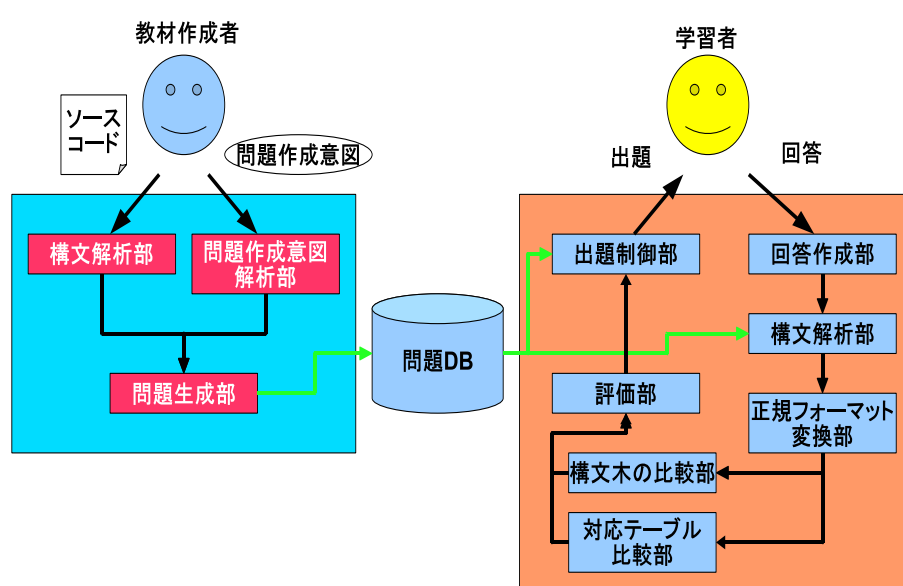


図 22: システムの流れ

7 実証実験

出題法の有効性を調べるために実証実験を行った。

7.1 実験方法

今回4章で述べた出題手法とランダムに出題する手法との比較を行い提案手法がどの程度学習効果があるのか調査を行った。

今回の実験は研究室の学部3年生から修士2年生の17名を対象に実証実験を行った。それぞれの被験者は一通りC言語について学習を済ませている人たちである。被験者を2グループに分け、提案手法の出題法で問題を解いてもらったグループとランダムに問題を出題するグループの2グループに分け実験を行った。このグループ分けでは、それぞれの出題手法において提案手法9名、ランダム手法8名に分けて実験を行った。実験内容は図23に示した流れにそって行った。

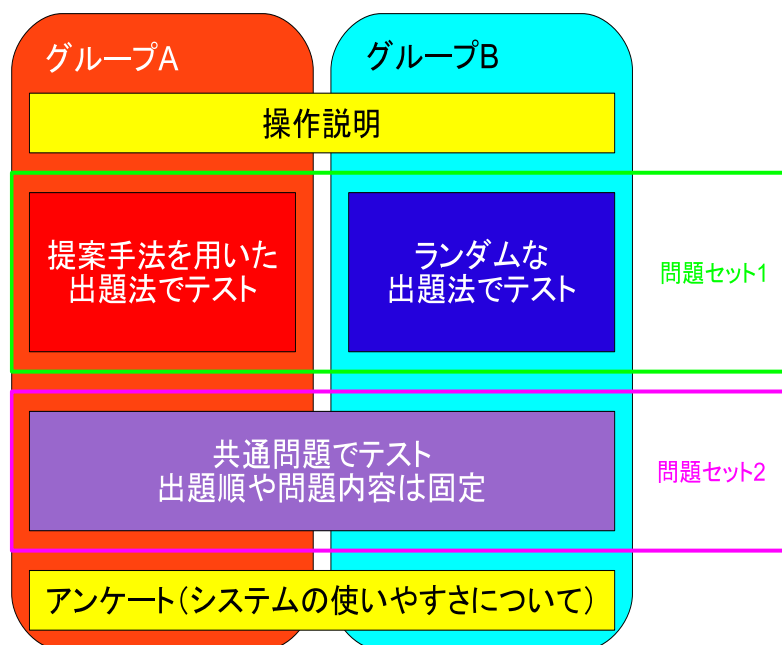


図 23: 実験の流れ

実験の流れはまずはじめにシステムの使い方の説明を行い、グループごとに提案手法とランダムでの問題指定方法のうちの決められた手法で問題を解いて

もらう。問題の内容は「ポインタと変数の動的確保」と「ポインタ配列と構造体ポインタ」の2種類の問題をそれぞれ7問ずつ解答してもらった。それぞれの問題は難易度ごとに3段階にレベル分けされているものを使用した。

続いてそれぞれの手法で問題を解いてもらった後、さらにどちらのグループにも同じ共通問題を解いてもらった。この共通問題は同一の問題を使用し、問題の出題順も同じになるように出題した。2回目のテストで用いた問題は1回目のテストで用いた問題より全体的に難易度が高くなるように設定してある。そして、最後にシステムの不満点などを問うアンケートを実施した。

その学習ログを解析して学習者ごとの正解率や正解数がどのように変化しているか調査を行った。

7.2 実験結果

今回実証実験を行い、受験者17名中15名分の有効データを取得することが出来た。取得したデータの内訳は手法ごとに提案手法8名、ランダム手法7名である。無効になった2名分のデータは途中でシステムエラーにより学習が続けられなくなったためである。データを取得できた2グループの被験者の人数構成を表5に示す。

表5: データが取得できた被験者の人数構成

学年	提案手法 [人]	ランダム手法 [人]
学部3年	2	2
学部4年	2	2
修士1年	3	3
修士2年	1	0

そして、その解答データを用いて解析を行い、出題手法の学習効果の調査を行った。今回の解析では1回目のテストと2回目に行ったテストの正解率を出し、それを各被験者ごとに1回目に比べて2回目のテストが「上がった」か「同じ」だったか、「下がった」かの3段階の評価を行った。また、各テストごとに被験者の正解数の平均値を手法ごとに出し、正解数の変化についても調査を行った。

まず、正解率の変化に着目した実験結果を示す。最初に表6に提案手法で学習を行った被験者の結果を示す。提案手法では(1)のポイントの分野と(1)と(2)の総合結果において2回目の方が正解率が上がった人が多かった。また、(2)のポイント配列の分野でも2回目の問題の難易度の方が高かったが半数以上の人が上がったまたは1回目と同じ正解率という結果になった。

表6: 提案手法

提案手法			
	上がった [人]	同じ [人]	下がった [人]
(1) ポインタ	4	2	2
(2) ポインタ配列	2	3	3
全体	4	2	2

同様にランダムな出題順で問題を提示したグループの結果を表7に示す。

表7: ランダムな出題順を用いた手法

ランダム			
	上がった [人]	同じ [人]	下がった [人]
(1) ポインタ	1	3	3
(2) ポインタ配列	0	0	7
全体	0	0	7

ランダムな出題順で問題を提示したグループでは、すべての分野において2回目に正答率が上がった人より、下がった人の方が多いという結果が得られた。また、(2) ポインタ配列の分野と総合結果においてはすべての人が1回目のテストより正解率が下がるという結果が得られた。

最後に、出題手法ごとに各分野における平均の正解率を表8に示す。また、この結果を用いてグラフを作成し図24に示す。

1回目の正解数は(1)のポイントの分野の方で多少差があるものの、ほぼ同じ正解数である。しかし、2回目の学習を見ると提案手法の正解数の方がランダムで出題したものより1問以上正解数が多いという結果になった。また、(1)の

ポイントの分野のように2問目の問題の方が難易度が高い問題にしたのに、平均の正解数が上がるという事例も出た。

表 8: 平均正解数

	提案手法 [点]	ランダム [点]	全体 [点]
(1) ポインタ「1 回目」	3.375	2.857	3.021
(1) ポインタ「2 回目」	3.875	2.429	3.215
(2) ポインタ配列「1 回目」	4.375	4.429	4.465
(2) ポインタ配列「2 回目」	3.125	2.000	2.674

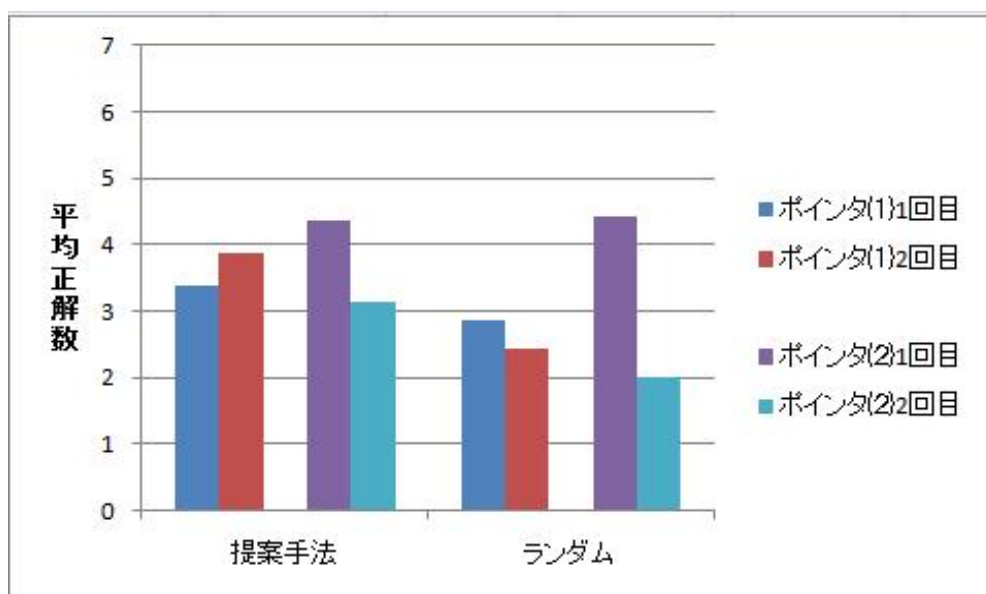


図 24: 平均正解数

7.3 考察

7.2 節で述べた実験結果からまず正解率の変化に着目すると、提案手法を用いてテストを受けたグループでは、2 回目の問題の内容が難しかったのにもかかわらず2 回目の正解率が上がったもしくは1 回目と同じだった人が半分以上を占めた。しかし、ランダムで出題したグループでは全体的に2 回目の正解率の方が1 回目の正解率より悪かった。以上の結果より、正解率の低下はランダムでの出題法に比べて正解率の低下した被験者の数が少なかったということが

分かった。

また、グループごとの平均正解数に着目すると、1回目のテストでは提案手法で出題したグループもランダムで出題したグループも平均正解率が同じくらいになっている。だが、2回目の平均正解数はそれぞれの出題分野において1問以上提案手法の方が高くなった。さらに、(1) のポイントの分野においては提案手法で1回目より2回目の方が学習者全体の成績が上がっていることが見て取れる。

よって、以上2点の実験結果から提案手法の方が学習効果が高いことが分かった。提案手法で問題を解いていくことによって学習を行うことが可能で、知識の定着に役立てることが出来る。

8 結論と今後の課題

本研究では、テストなどでの問題作成者の負荷を減らすため、C言語学習を対象とした穴埋め問題の自動生成機能の開発を行った。また、生成した問題を用いて出題や採点を行い、繰り返し学習を行うことが出来るシステムの提案と実装を行った。そのうえで、出題手法について、提案した手法とランダムで問題を出題した手法とどちらが学習効果があるのか実証実験を行い評価をした。

これまで問題作成者は問題作成時に一から問題を考えPCに入力したり、アルゴリズムの記述やデータベースへの登録など多くの負荷がかかっていた。今回提案・実装を行った問題の自動生成機能により、ソースコードを登録し、登録された問題作成意図を選択するだけで問題候補を生成することが出来る。さらに、問題候補の中から出題したい問題を選択すると自動的にデータベースに登録することが可能なため、複雑なアルゴリズムの記述などを行う手間が省け問題作成の負荷を軽減できる。さらに、出題機構を用いると学習を行わせるためにルールの記述等を行わなくても、適応型の出題が可能となり問題作成者の負荷が軽減出来ると考えられる。

採点においても、これまでは教師が臨機応変に学習者のプログラミングスタイルに合わせて判断を行い採点してきた。しかし、これまでの方法では大量の学習者がいた場合、採点にも多くの負荷がかかっていた。本研究では、構文木の比較や正規フォーマットを用いて自動で採点を行えるようにすることで、採点にかかる教師への負荷が大幅に軽減できると考える。また、解答したその場で自動採点を行うことが出来るため、採点結果を踏まえすぐに次の出題時の問題切り替えるという適応型の学習が可能となる。

今後の課題としては以下のものがあげられる。

- 自動生成した問題の有用性の評価

今回は時間の都合上出来なかったが、提案した問題の自動生成で作成された問題が、学習に用いることが出来るレベルなのかどうか評価を行う必要がある。さらに、問題作成意図により問題作成者の思い通りの問題が作られているかどうかの検証も行う必要がある。よって、演習などを担当する先生方にシステムを使って問題を生成してもらいアンケートを実施するなどの実証実験を行う必要がある。

- システム全体の有用性の評価

今回は出題法にのみ有用性を示す実証実験を行ったが、自動生成から出題、採点すべてを通しての実証実験を行っておらず、穴埋め問題の自動生成や採点などの定量的な評価が出来ていない。多くの人にシステムを使ってもらいシステムの評価を行う必要がある。

- 採点部分の実装

本論文で提案した採点部分が構文木の比較部分と正規フォーマットの変換部の一部（while 文と for 文，分配律）しか実装されておらず，それぞれの結合テストも出来ていない。よって，現状では採点は学習者が答えた解答と模範解答の全文一致での採点を行っているので，未実装部分を早急に実現する必要がある。

- 正規フォーマットの拡張

採点時より幅広い学習者のプログラミングスタイルに対応するために，様々な解答のパターンに対応できるように正規フォーマットを拡張していく必要がある。

謝辞

本研究を進めるにあたり、懇切な御指導、御鞭撻を頂いた横田一正教授に深く感謝します。

また、本研究に関し多くの有益な御助言、御助力を頂いた國島丈生准教授に感謝致します。

最後に日頃から有意義な御助言、御協力を頂いた知能メディア工学研究室の諸氏に厚く御礼申し上げます。

参考文献

- [1] 萩原秀和, 富永浩之, 松原行宏, 山崎敏範, ”ドリル練習に基づく Web 型試験システム-情報基礎科目への応用と運用試験-, ” 信学技報, vol.102, no.388, ET2002, pp.57-62, 2002.
- [2] 萩原秀和, 富永浩之, 松原行宏, 山崎敏範, ”個に対応するドリル型 CAI システム-学習者レベルに適応する問題提示-, ” 信学技報, vol.103, no.697, ET2003, pp.197-202, 2003.
- [3] 玉木久夫, 疋田輝雄, 井口幸洋, 小島崇輝, 早川智一, ”MAX/C: WEB 上の C プログラミング実習システム, ” 全国大学 IT 活用教育方法研究発表会, E-10.
- [4] 内田保雄, ”初級プログラミング学習のための自動作問システム, ” 情報処理学会研究報告, Vol.2007, No.123(20071207) pp. 109-113 2007-CE-92-(16).
- [5] 新開純子, 早瀬欣和, 宮地功, ”Moodle を基盤としたプログラミング教育のための穴埋め問題生成に関する検討, ” 信学技報, vol.108, no.247, ET2008, pp.5-10, 2008.
- [6] 池田絵里, ”C 言語学習のための穴埋め問題自動生成システム, ” 岡山県立大学 卒業論文, 2008.
- [7] 有安浩平, 池田絵里, 岡本辰夫, 國島丈生, 横田一正, ”学習者に合わせた C 言語演習穴埋め問題の自動生成, ” 第 1 回データ工学と情報マネジメントに関するフォーラム (DEIM2009), D9-5, 2009.
- [8] 西輝之, 延原哲也, 劉渤江, 横田一正. ”適応型 e ラーニングに必要な診断的テスト機構”. 夏のデータベースワークショップ 2006 (DBWS2006), A1-4, 2006.
- [9] ANTLR. ANTLR Parser Generator. <http://www.antlr.org/>.
- [10] JAVA. <http://java.sun.com/>.
- [11] PostgreSQL. <http://www.postgresql.org/>.
- [12] JDBC. <http://java.sun.com/javase/technologies/database/>