プログラミング学習における構文図式を用いた構文理解支援方法の提案

安達 有希[1,a)]　蜂巣 吉成[2,b)]　吉田 敦[2,c)]　桑原 寛明[2,d)]　阿草 清滋[3,e)]

概要：プログラミング学習において，学習者の構文理解を支援するために，構文図式の非終端記号のノードに学習者が記述したプログラム断片を記述した図式 (構文図式インスタンスと呼ぶ) を提案する．構文図式と構文図式インスタンスを表示するツールを試作し，構文の誤りによるコンパイルエラーや期待する実行結果を得られないプログラムなどに利用することで学習者の構文理解を支援する．実際にツールを学習者に使用してもらい，その評価を行った．

キーワード：プログラミング学習，構文理解，構文図式

A Support System for Understanding the Syntax of Programming Language based on a Syntax Diagram

Yuki Adachi[1,a)]　Yoshinari Hachisu[2,b)]　Atsushi Yoshida[2,c)]　Hiroaki Kuwabara[2,d)]
Kiyoshi Agusa[3,e)]

Abstract: We propose a syntax diagram instance, which shows the corresponding program fragments in non-terminal nodes of a syntax diagram, in order to help learners to understand the syntax of a programming language. We have implemented a tool showing syntax diagrams and syntax diagram instances. The tool helps learners to understand the syntax errors and the reasons for the unexpected execution results of their own programs. We have evaluated our approach by using the tool in a programming exercise of a student group.

Keywords: Programming Education, Understanding Syntax, Syntax Diagram

1. はじめに

大学などでのプログラミング教育では，条件分岐や繰返しなどの概念を教える際に，プログラミング言語の構文と意味，それに従って記述されたプログラム例を示すことが一般的である．学習者，特に初めてプログラミング言語を学ぶような学習者は，構文とその意味を理解してトップダウン的にプログラムを作成していくことは難しい．プログラム例を参考にして自身でプログラムを作成しながらボトムアップ的に構文を理解していくことが多いが，構文について意識しない学習者もいる．

筆頭著者がプログラミング言語を学習してきた中で，難しいと感じたのがコンパイルエラーとなるプログラムの理解や修正であった．何がどう間違っているのかを理解することを難しく感じていたが，エラーの多くが構文の間違いであり，構文の理解をしていないことが原因であることがわかった．構文を意識せず理解もしていないまま，講義資

[1] 南山大学 大学院理工学研究科 Graduate School of Science and Engineering, Nanzan University, 18 Yamazato-cho, Showa-ku, Nagoya-shi, 466-8673, Japan
[2] 南山大学 Nanzan University, 18 Yamazato-cho, Showa-ku, Nagoya-shi, 466-8673, Japan
[3] 京都高度技術研究所 ASTEM, 134, Chudouji Minamimachi, Shimogyou, Kyoto, 600-8813, Japan
[a)] m18se001@nanzan-u.ac.jp
[b)] hachisu@nanzan-u.ac.jp
[c)] atsu@nanzan-u.ac.jp
[d)] kuwabara@nanzan-u.ac.jp
[e)] agusa@astem.or.jp

料や教科書を真似てプログラムを書くだけでは，プログラムの間違いにも気がつきにくく，応用も利かない．そのような学習者に構文を意識させ理解を促すことができれば，自身が書いたプログラムをより深く理解でき，構造が類似した構文間や他言語での構文を学ぶための抽象化能力を養うこともできる．

本研究では，プログラミング学習において，学習者の構文理解を支援するために，構文図式の非終端記号のノードに学習者が記述したプログラム断片を記述した図式 (構文図式インスタンスと呼ぶ) を提案する．また，構文図式と構文図式インスタンスを表示するツールを試作し，実際に使用してもらうことで評価を行った．このツールは，構文誤りによるコンパイルエラーが修正できない，期待する実行結果を得られない，作成したプログラムの構文を確認したいといった場面で学習者が利用することで，構文理解を支援する．

2. 理解支援方法の提案

2.1 構文理解の重要性

本研究を進めていくにあたり，指導者と学習者の認識の違いがあることがわかった．指導者がプログラミングを教える際，講義資料や教科書を利用する．それらを利用し，学習する側が構文を意識して学習していくと考えていることが多いようだが，学習者はその意図を理解していない場合がある．与えられた講義資料や教科書に載っているプログラムをそのまま書き写したり，一部だけを書き換えたりし，構文に意識を向けることなく講義資料や教科書を利用する学習者も多い．

構文を理解していないとコンパイルエラーの修正が難しい場合もある．Listing 1 は構文誤りがあるプログラムである．gcc (Version 4.6) [1] でコンパイルを行うと図 1 のようなエラーメッセージが表示される．else の前に if がないことを指摘しているが，プログラム中には if が記述されており，何が誤っているのか気づきにくい．clang (Version 3.9) では，図 2 のようなエラーメッセージが表示される．clang は gcc よりも詳細で理解しやすいメッセージが表示されると言われているが [2]，この場合は誤りの理由がわかりにくい．また，学習者の中には，構文における式 (expression) を理解していない場合もある．構文を理解していない学習者にとっては，これらのエラーメッセージを頼りにプログラムの修正を行うことは難しい．

Listing 2 は自然数を素因数分解するプログラム例であるが，for 文のループ本体が空文となっており，コンパイルができても期待する結果は得られない．構文自体は間違っていないので，実行するまで間違いに気付きにくい．

Listing 3 と Listing 4 は num が 0 か判定するプログラム例で，どちらも正常にコンパイルでき，期待する実行結果が得られる．構文を理解していなければ，なぜ書き方が異なっているにもかかわらず同じ結果になるのかは理解ができない．

これらは講義資料や教科書を構文にも着目しながら読んでいれば，理由を理解することができる．また，構文を意識することで構文間で構造が類似している部分やそうでない部分に注目でき，理解も深まりやすくなる．構文を理解していれば他言語にも応用が利き，構文を学ぶための抽象化能力を養うこともできる．

Listing 1 if-else に誤りのあるプログラム例

```
if (n%i == 0)
    printf("%d", i);
    n = n/i;
else
    i++;
```

```
fac.c: 関数 ‘main’ 内:
fac.c:14:3: エラー: ‘else’ の前に ‘if’ がありません
```

図 1 Listing 1 の gcc のコンパイル結果

```
fac.c:14:3: error: expected expression
                else
                ^
1 error generated.
```

図 2 Listing 1 の clang のコンパイル結果

Listing 2 for が空文のプログラム例

```
for (i=2; n>1;);
    if (n%i == 0){
       printf("%d", i);
       n = n/i;
    }else
       i++;
```

Listing 3 if-else が複合文のプログラム例

```
if (num){
    puts("The number is not zero.");
}else{
    puts("The number is zero.");
}
```

Listing 4 if-else が単文のプログラム例

```
if (num)
    puts("The number is not zero.");
else
    puts("The number is zero.");
```

2.2 想定する理解支援の場面

本研究では，学習者が次の場面で提案ツールを利用することで，構文の理解支援を行うことを想定している．

- コンパイルエラーを修正できない
- 期待する実行結果を得られない
- 作成したプログラムの構文を確認する

コンパイルエラーとなる原因として，特に多いのが構文の誤りである．しかし，構文の誤りにより表示されるエラーメッセージは，構文を理解していない学習者にとって，わかりにくいものである．

コンパイルは可能だが期待する実行結果とは異なる場合，その原因を見つけることは困難である．中には，構文に起因するものも存在する．構文を理解することで，そのような誤りに気づくことができる．

コンパイル可能な誤りのないプログラムを書いている学習者の中でも，構文を理解できていない学習者が存在する．そのような学習者にも，自分の書いたプログラムの構文図式インスタンスをツールで確認させることで，構文を意識させ，理解を促すことにつながる．

2.3 提案方法の概要

本研究では，構文を構文図式で，プログラムを構文図式インスタンスで可視化する記法を提案し，それらを表示するためのツールの試作を行う．

構文を表す方法には BNF や構文図式があるが，BNF はテキストであるので，構文図式のように図で表す方が学習者にとってわかりやすいと考えた．プログラムを図で表す方法には PAD などの構造化フローチャートもある．プログラムの制御の流れを表すには適しているが，必ずしも構文の理解にはつながらないと考える．

プログラムの構文を可視化する方法として構文木があるが，構文木はプログラムを構文解析した結果を木構造で表したものであり，学習者が構文規則の詳細を理解していないと，なぜそのような木構造になるのかがわからない．構文図式インスタンスは構文を正確に理解していなくても，ある程度読むことができ，何度も読むことによって構文を理解できる．

本研究で扱う構文は，if 文，while 文，for 文である．これらは演習においてよく利用される文であり，学習者がこれらの構文を利用する機会が多いと判断した．構文理解を促すのに適していると考える．

2.4 構文図式

本研究では，文献 [3] を参考に，if 文，if-else 文，while 文，for 文をそれぞれ図 3，図 4，図 5，図 6 の構文図式を用いて表す．終端記号を丸囲みで表し，if や while などの構文の種類を表す予約語は二重丸囲みで表している．式や文などの非終端記号は四角囲みで表している．

構文図式の各ノードには，IF や i_lp などの名前が付いており，それぞれ図 3，図 4，図 5，図 6 の丸囲みや四角囲みのノードの下に表示している通りである．

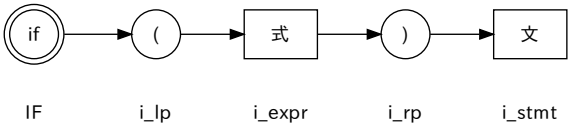


図 3 if の構文図式

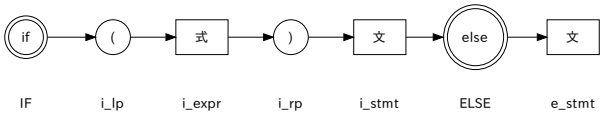


図 4 if-else の構文図式

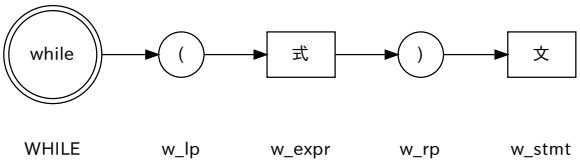


図 5 while の構文図式

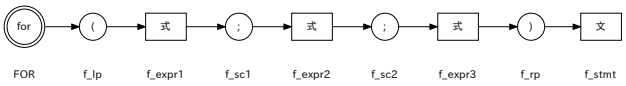


図 6 for の構文図式

2.5 構文図式インスタンス

本研究では，図 7 のように構文図式の非終端記号を表す四角囲みのノードにそのノードに対応するプログラム断片を記述した図式を構文図式インスタンスと呼ぶ．

学習者の書いたプログラムに欠如している部分があれば，ノードにはプログラム断片ではなくノードの名前を入れる (図 8)．ただし，for 文の式の部分が空式になっている場合は欠如ではないので，ノード名は入らない (図 14)．

図 5 のような構文図式と図 7 や図 8 のような構文図式インスタンスを上下に並べて表示することで，学習者は自身が書いたプログラムの構文がどのような構造になっているのかを比較して確認することができる．

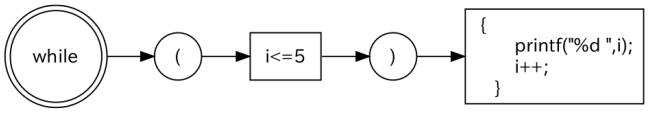


図 7　構文図式インスタンス

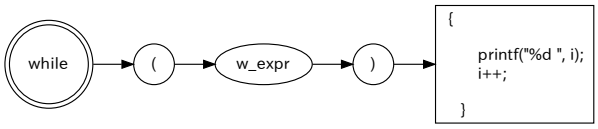


図 8　式が欠如したプログラムの構文図式インスタンス

2.6　ツールの設計と実現

ここでは，本研究で試作した図式を表示するツールについて述べる[*1]．このツールは図式生成ツールと HTML 生成ツールの 2 つから構成される．

図式生成ツールは，入力されたソースプログラムを TEBA[4] を用いて構文解析を行う．構文解析の対象となるプログラムは，構文エラーを含む可能性があるので，通常の構文解析の方法は利用できない．そこで，本研究では構文エラーも許容しつつ構文解析をし，近似的な構文木を生成する TEBA を採用した．TEBA では if 文や while 文の制御式が空である場合や for 文の () 内の; が足らない場合なども近似的に解析可能である．

TEBA による構文解析により，プログラム中に出現する if 文，while 文，for 文の構文図式と構文図式インスタンスの画像ファイルを Graphviz[5] を利用して出力する．

文献 [3] に示されているサンプルプログラム 205 編のうち，if 文，while 文，for 文の 3 種類の構文が含まれている 134 編に対して図式生成ツールを使用したところ，すべてのプログラムで適切な図式の生成を行うことができた．

HTML 生成ツールはソースプログラムと図式生成ツールで出力された画像ファイルを入力とし，プログラムの if 文，while 文，for 文の予約語部分に画像を表示するためのリンクを含んだ HTML ファイルを出力する．

例えば，HTML ファイルは図 9 のようにブラウザで表示される．学習者がリンクをクリックすると図 10 や図 11 のように図式がポップアップで表示される．

2.7　期待される効果

図式の生成は構文ごとにそれぞれ行われ，学習者の書いたプログラムが，ほぼそのままの形で構文図式インスタンスに記述されるので，複数の入れ子の複雑なプログラムでも，自分の書いたプログラムの構文の構造がどうなっているかがわかる．

構文図式インスタンスだけでなく，その構文にあった構文図式も同時に表示することによって，学習者が構文と自分の書いたプログラムの構文を見比べ，確認することができ，構文を理解していない学習者だけでなく，講義資料や教科書を真似ているだけの構文を意識していない学習者にも，構文を学ばせるのに有効である．

構文部分に誤りのあるプログラムからも，構文ごとにそれぞれ図式の生成が行われる．本研究のツールを使用することによって，誤りの有無に関わらず，どのようなプログラムからでも if 文，while 文，for 文の構文ならば，1 つの構文に対して 1 つの図式を生成できる．誤りのあるプログラムからも図式の生成が行えることによって，構文に起因するような誤りに気づくことができると考える．

[*1] https://akg.se.nanzan-u.ac.jp/syndia/ で公開している．

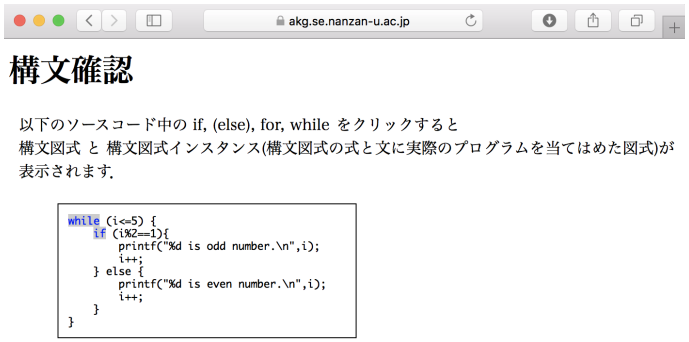


図 9　ブラウザでのプログラムの表示

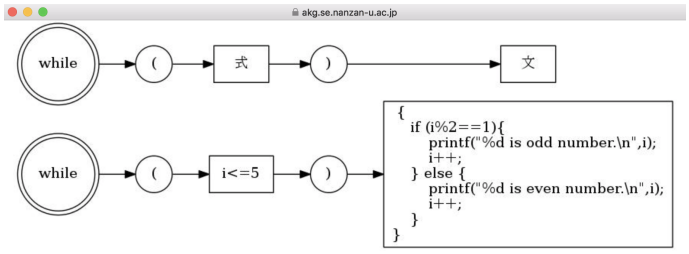


図 10　図 9 の while の図式をポップアップで表示

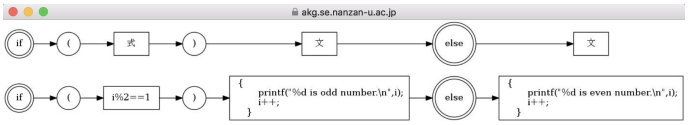


図 11　図 9 の if の図式をポップアップで表示

2.7.1　コンパイルエラーの場合

2.1 節で述べた Listing 1 のような誤りを含んだプログラムからは，図 12，図 13 の図式が生成される．図 12 を見れば 1 つの文しか if 文の文として認識されていないので複数の文を複合文にし，1 つの文として記述する必要があることがわかる．TEBA は，予約語 else に対して，それに対応する予約語 if がない場合，if を伴なわない単体の else 文を構成するので，図 13 の図式も生成できる．図 12 だけでなく図 13 も生成することによって，これらの図から誤りの原因に気づくことができるだけでなく，なぜこのようなエ

ラーメッセージが表示されるのかもわかる．

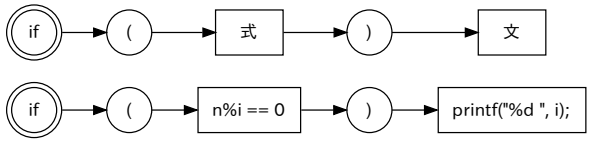


図 12 Listing 1 の誤りがある例の図式 1

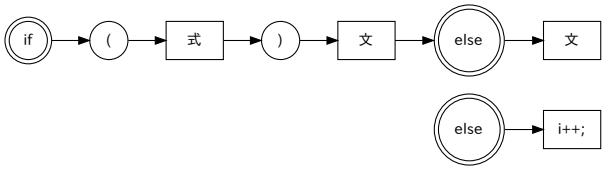


図 13 Listing 1 の誤りがある例の図式 2

2.7.2 期待する実行結果と異なる場合

Listing 2 のような実行可能であるが，期待する実行結果と異なるプログラムも，提案図式を用いた図 14 を見れば本来繰り返したい文が空文となっていることがその原因であることがわかる．

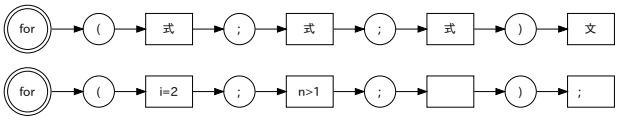


図 14 Listing 2 の空文の例の図式

2.7.3 期待する実行結果通りの場合

2.1 節で述べた Listing 3 と Listing 4 のような書き方が異なっていても同じ実行結果を返すプログラムも，それぞれ提案図式を用いた図 15，図 16 を見ることによって，文が複合文の場合でも単文の場合でも，どちらも同じ文であることがわかる．

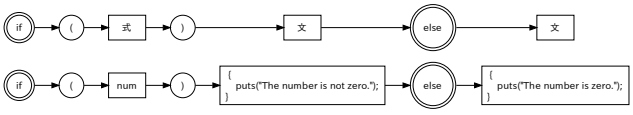


図 15 Listing 3 の文が複合文の例の図式

Listing 5 は 3 変数の中央値を求めるプログラムの一部分である．ある条件による分岐が複数ある場合，4 行目と 11 行目のように else と if を同一の行に書くことが多いが，

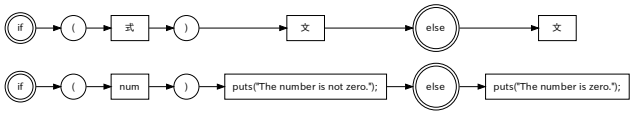


図 16 Listing 4 の文が単文の例の図式

構文を理解していない学習者は，2 行目と 9 行目の if-else 文は if-else if-else 文のような誤った構文で理解している場合がある．構文図式インスタンスでは図 17 や図 18 のように，if-else 文として表示されるので，正しい構文理解を支援できる．

Listing 5 3 変数の中央値の例

```
if (x>=y)
    if (y>=z)          /* 2行目 */
        m=y;
    else if (x>z)      /* 4行目 */
        m = z;
    else
        m = x;
else
    if (x>=z)          /* 9行目 */
        m = x;
    else if (y>=z)     /* 11行目 */
        m = z;
    else
        m = y;
```

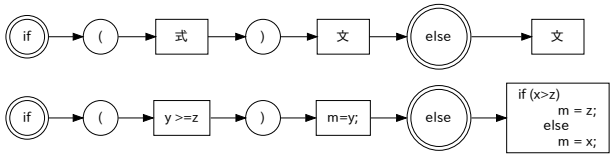


図 17 3 変数の中央値の例の図式 1

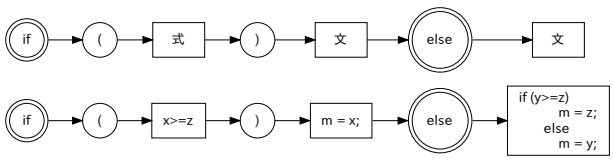


図 18 3 変数の中央値の例の図式 2

Listing 6 は，ぶら下がり else のプログラムの一部分である．図 19 と図 20 の構文図式インスタンスにより，else 文がどちらの if 文と対応しているかがわかる．

Listing 6 ぶら下がり else の例

```
if (a > 0)
if (b > 0)
    printf("X\n");
else
    printf("Y\n");
```

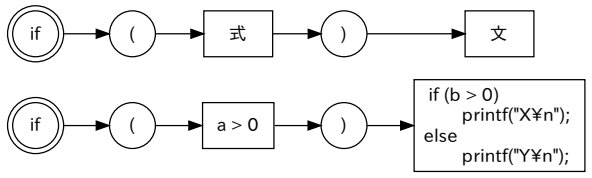


図 19 ぶら下がり else の例の図式 1

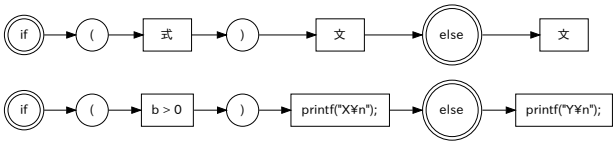


図 20 ぶら下がり else の例の図式 2

3. 評価

3.1 評価方法

プログラミングの講義および実習で，C 言語を一通り学習した学部 3 年生 10 名を対象に構文理解に関する演習を行った．

比較を行うために，学生を A と B の 2 グループに分けた．グループ間の能力差を減らすために，図 21 のアンケートの回答に基づいてグループ分けをした．

次に，本研究で提案するツールの効果を調べるために，図 22 の問題に解答してもらった．A グループはツールを利用せず，B グループはツールを利用して問題に解答した．

3.2 グループ分け

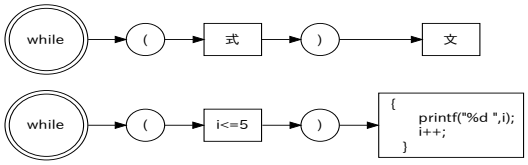
表 1 アンケート問 1 の回答結果

	A グループ	B グループ
1	0 人	1 人
2	1 人	3 人
3	2 人	1 人
4	2 人	0 人
5	0 人	0 人

表 1，表 2 はそれぞれアンケートの問 1 と問 2 の各グループの回答結果である．問 1 の結果から演習時に構文を意識している学生は少ないものの，問 2 の結果から構文があることは知っていることがわかる．

1. プログラミングの演習や課題を行う際に講義資料や教科書を用いるとき，最も注目するのはどこですか．
（1） 構文の説明 （2）プログラム例（ソースコード）
（3） プログラムの説明 （4） 実行例，実行結果 （5） その他
2. プログラミング言語の構文に対して当てはまるものはどれですか．
(1）構文を理解している（どのような構文があるか説明できる）
(2）構文を知っている（構文があることを知っている）
(3）聞いたことがある（何かはわからないが，聞いたことはある）
(4）全く分からない（聞いたこともない）
3. 次の while のプログラム例を構文図式 (図上) に当てはめると，図下のようになります．

```
while (i<=5) {
    printf("%d ",i);
    i++;
}
```



これを参考に if による条件分岐の構文図式と，次のプログラムを構文図式に当てはめた図式を書いてください．複数の図式になっても構いません．

```
if (n == 0)
    printf("n is 0");
else if (n > 0)
    printf("n is greater then 0");
else
    printf("n is less than 0");
```

図 21 アンケート設問

4. Listing 6 のぶら下がりの else について
4-1. 実行せずに a, b の正負 4 通りの出力を記述する．
4-2. 実行して a, b の正負 4 通りの出力を確認して，そのような結果になる理由を述べる．
4-3. 次の表の出力になるようにプログラムを修正する．

a	b	出力
1	1	X
1	-1	なし
-1	1	Y
-1	-1	Y

5. Listing 1 の素因数分解のプログラムを意図した結果になるように修正する (コンパイル，実行してよい)．

図 22 演習に用いた問題

問 3 の結果では，if-else 文の構文図式を正しく記述できた学生はいなかった．else のない if 文の構文図式（図 3）を記述できた学生は 2 人いたが (2 人とも A グループ)，どちらの学生も if と else を別々の構文図式として記述していた．このことから，プログラミングを学んでいても，構文を正確に理解してはいないようである．

表 2 アンケート問 2 の回答結果

	A グループ	B グループ
1	0 人	0 人
2	4 人	4 人
3	1 人	0 人
4	0 人	1 人

2.7.3 節で述べたように if-else if-else 文として図 23 のように記述した学生は 2 人いた (2 人とも A グループ).

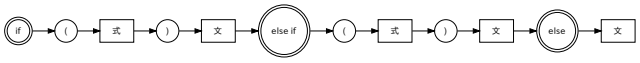


図 23 連続する 2 つの if-else 文の記述例 1

if から分岐している図 24 のような記述をした学生が 7 人おり (A グループに 3 人, B グループに 4 人)，構文と実行における制御フローを混同している可能性がある.

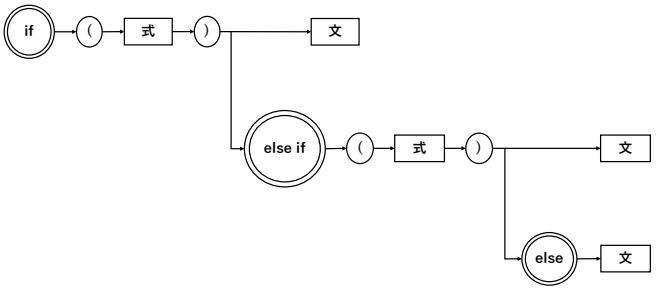


図 24 連続する 2 つの if-else 文の記述例 2

「式」や「文」と記述する構文図式を記述できた学生は 6 人 (A グループに 4 人, B グループに 2 人) である．構文図式を記述せずに，構文図式インスタンスのみの記述をした 4 人 (A グループに 1 人, B グループに 3 人) は構文図式と構文図式インスタンスの区別がついていないようである.

グループ分けは，演習中にアンケートの問 1-3 の回答を回収した後に，これらの分析を行う前に短時間で行った．同じ能力になるように調整をしたが，結果的に A グループの方が構文に関して知識があるグループ分けになったようである.

3.3 演習結果

表 3 は図 22 の問題の解答結果をまとめたものである．問 4(1) と問 4(2) は，ツールの使用はさせず，どちらのグループも同じ条件で解いてもらい，問 4(3) と問 5 は，B グループのみツールを使用してもよいことにして解いてもらった.

問 4(1) の結果から，ぶら下がり else がどのように構文解析されるかわかっていない学生もいた．うち 1 人 (B グループ) は，出力の Y を F と書いており，問題を読み間違えたと考えられる.

表 3 図 22 の解答結果

		A グループ	B グループ
4(1)	正	3 人	3 人
	誤	2 人	2 人
4(2)	正	4 人	3 人
	誤	1 人	2 人
4(3)	正	5 人	3 人
	誤	0 人	2 人
5	正	4 人	4 人
	誤	1 人	1 人

問 4(2) では A グループの 1 人が「適切な位置に {} がない」と解答していた．問題では期待される出力を示して，その出力が得られない理由を聞いているわけではないので，誤りとした．B グループの誤り 2 人は未解答であった.

問 4(3) を間違えた学生のうち 1 人は，適切に {} をつけることができていたが，else の文を `if(a<-1)` の if 文として書いていた．もう 1 人の学生は if の条件式の `(a>0)` と `(b>0)` を入れ替えただけであった.

問 5 のプログラムの誤りを修正する問題は，修正箇所は 2 箇所である．不正解のうち，A グループの学生は，そのどちらも不正解であった．B グループの学生は，修正すべき箇所は 2 箇所とも修正できていたが，else を削除して `i++`を for 文の 3 番目の式とする誤った修正をしていた.

3.4 評価

今回行なった演習では，ツール利用なしの A グループの方が正解者が多かったが，3.2 節で述べたように A グループの方が元々構文の理解が深かったようである．3 年生なので，今回の問題のエラーはこれまでにも経験していると考えられる.

問 3 の回答結果から構文理解が必要であることや問 4(1) のぶら下がり else がどのように構文解析されるかわかっていない学生もいたので，ツールによる構文理解支援は有効であると考える.

B グループは全員が，問 5 の問題の期待する実行結果が得られない空文の誤りを修正できている.

ツールなしの方は最小限の位置以外にも {} をつけていることが多く，ツールありの方は最小限の位置にのみ {} をつけていることが多い．{} を付け加える問題は問 4(3) と問 5 の 2 問である．問 4(3) は A グループが 5 人中 2 人，B グループが 3 人中 2 人，問 5 は A グループが 4 人中 0 人，B グループが 4 人中 2 人が最小限の位置にのみ {} をつけていた．提案ツールにより構文を視覚的に確認することで，必要なところのみに {} を入れたと考えられる．このことから，2.7 節で述べた期待される効果があったと考

える．

演習では最後にツールの感想も記述してもらった (A グループには解答後，ツールを利用してもらった) ．ツールを使うと視覚的にわかりやすく頭の中が整理されたなど，7 人が好意的な意見だった．1 人は自分で構文を予測した後に確認用にツールを使ったと回答した．2.2 節で述べた，作成したプログラムの構文の確認用に使っていた．1 人は今度はツールを使ってみたいとの回答だった．A グループの学生だが，時間がなくツールを利用できなかったようである．1 人は無回答だった．

好意的な回答をした学生の中には，括弧が多くなると図式が見にくくなるのではないか，1 つしか画像が表示できないのが不便などの意見もあった．前者は構文図式インスタンスの文のプログラム部分が多くなるという指摘，後者は if 文，while 文，for 文の構文ごとでなく，プログラム中のすべての if 文，while 文，for 文の構文に対する図式を 1 つの画像で表示したいという要望だと考えられる．

図式の表示について，ツール開発の初期段階では 1 つのプログラムから生成される図式をすべてまとめた 1 つの画像として表示していた．この表示方法では構文が多くなると，どの図式がどの構文と対応しているのかがわかりにくいので，1 つずつ表示させる方法に変更した．プログラムの入れ子になっている部分を 1 つにまとめた画像として表示するなどの改善の余地がある．

4. 関連研究

岩崎らの研究 [6] はコンパイラの内部の処理状況を，機能単位ごとに分けて考え，構文図式や構文木などを用いて可視化を行っている．構文解析については，再帰下降パーサが処理している構文図式を次々と重ねて表示し，その動作を可視化している．コンパイラの動作理解を目的としており，本研究が対象とするプログラミング学習者の構文理解には向かない．

Avis[7] は，プログラムの読解支援を目的に，Java のソースコードを解析して，プログラムの流れを理解するためのフローチャートや，プログラム実行時の振舞いを理解するための逐次型実行経路図などを出力する．構文を理解させるための支援は行っていない．

HelpMeOut[8] は，エラーの解決方法をクラウドで共有し，エラーが起きた時に適切な修正方法を提案するツールである．ソースコード中のエラーのある行を選択し，HelpMeOut を参照すると，提案パネルが表示され，いくつかの解決策が表示される．解決策にはそれぞれ詳細な情報を表示したり正しい記述をそのままコピーしてエラーを直すボタンがある．HelpMeOut は，時間の経過とともに変更されるソースコードのログをデータベースへ収集している．そのログを基に，提案パネルはエラーに対する解決策を示している．データベースから修正の優先順位リストを解決策を表示している画面へ表示し，新しく追加された修正の説明をデータベースへ送信することで解説を集めている．修正方法を提示するが，ただその通りにプログラムを修正していると，必ずしも構文を理解することには繋がらない．

5. おわりに

本研究では，C プログラムの構文理解支援を行うために，構文図式インスタンスを提案し，構文図式とともに表示させるツールの試作を行い，学習者に視覚的に構文を意識させ，理解を促せるようにした．演習において，提案方法が想定される場面で利用されたこと，期待される効果があったことから，その有効性を確認した．

本研究で扱った構文は 3 つだが，この他にも構文を増やしていくこと，より多くの学習者に使ってもらい評価をすることが今後の課題である．近年では Scratch や Google Blockly のようなブロックを用いたビジュアルプログラミング言語も普及しているので，自分の書いたプログラムをブロックを用いて図示する方法も今後検討していきたい．

謝辞 卒業研究として一緒に取り組んだ牧野美里さんと小原友貴さん，演習に協力してくれた 3 年生に感謝する．

本研究の一部は，JSPS 科研費 17K00114，17K01154，2018 年度南山大学パッヘ奨励金 I-A-2 の助成を受けた．

参考文献

[1] 公式サイト：GCC -GCC, the GNU Compiler Collection，https://gcc.gnu.org

[2] 公式サイト：Clang - Expressive Diagnostics，https://clang.llvm.org/diagnostics.html

[3] 柴田望洋：新・明解 C 言語 入門編，SB クリエイティブ株式会社 (2014).

[4] 吉田敦，蜂巣吉成，沢田篤史，張漢明，野呂昌満：“属性付き字句系列に基づくソースコード書き換え支援環境”，情報処理学会論文誌，Vol.53 No.7，pp.1832-1849(2012).

[5] 公式サイト：Graphviz - Graph Visualization Software，http://www.graphviz.org/

[6] 岩崎克治，辻野嘉宏，都倉信樹：“教育を目的としたコンパイラの動作の可視化手法とその効果について”，情報処理学会研究報告コンピュータと教育 (CE)，1990-CE-011，pp.1-8(1990).

[7] 喜多義弘，片山徹郎，冨田重幸：“Java プログラム読解支援のためのプログラム自動可視化ツール Avis の実装と評価”，電子情報通信学会論文誌 D，Vol.J95-D No.4，pp.855-869(2012).

[8] Björn Hartmann，Daniel MacDougall，Joel Brandt，Scott R. Klemmer：“What Would Other Programmers Do?Suggesting Solutions to Error Messages”，CHI 2010:Understanding and Supporting Programming，pp.1019-1020 (April 10-15，2010，Atlanta，GA，USA)