C 言語プログラミング学習における 制御文を中心とした要約生成ツールの提案

2016SE045 栗田裕生　2016SE083 富田佑太郎
指導教員：蜂巣吉成

1　はじめに

プログラミング学習において，プログラムを理解することは重要である．プログラムを理解していないと，期待していない実行結果になった場合に修正が困難であったり，偶然期待した実行結果が得られた場合に誤りに気づけない．しかし，次に示す理由により学習者にとってプログラムを理解することが難しい場合がある．

1. プログラムのソースコードがプログラミング言語特有の字句や文法によって記述されている．
2. 手続き型言語において代入や分岐，繰り返しの組合せで構成される処理を理解する必要がある．

プログラムの理解支援としてソースコードを自然言語で表現する手法が存在する．その一つとしてふりがなプログラミング [1] が提案されている．ふりがなプログラミングはソースコードにふりがなを振る，また読み下し文を用いることで理解支援を行う．しかし，この手法は問題点 1 は支援するが，問題点 2 に対して支援していない．例えば，配列の要素の合計を求めるプログラムに適用した場合，「変数 sum に 0 を代入する．変数 i に 0 を代入し，継続条件 ”変数 i が 5 より下の間” が真の間以下を繰り返す．変数 sum と配列 a[i] の和を変数 sum に代入する．」という読み下し文となり，「配列 a[0]〜a[4] の合計を求める．」という計算処理が理解しづらい．

本研究では，制御文を中心とした C 言語ソースコード要約生成ツールを提案する．学習用なので典型的な問題についてのパターンを用意して，学習者が記述した C 言語ソースコードがパターンにマッチングした場合に，それに対応する要約を生成する．予めすべての問題のパターンを用意することは難しいので，本ツールを拡張可能な仕組みとした．生成する要約は分岐や繰り返しの制御文を中心とした一連の計算を簡潔に表現したものであり，誤りを含んだプログラムの場合，学習者に誤りの発見を促せるものとする．そのために学習者が犯しやすい誤りを分類し，要約を整理した．本ツールによって期待した実行結果が得られるソースコードを記述できた学習者は記述したソースコードの内容を再確認することができ，記述できなかった学習者は，記述したソースコードによって行われる処理を理解した上で，誤りに気づくことができる．

配列を用いた繰り返しが扱えれば分岐や繰り返しも扱えるので，本研究で対象とするソースコードは新・明解 C 言語 入門編 [5] に掲載されている 5 章配列と 9 章文字列の例題のうち，制御文が含まれているものとする．またコンパイル時のエラーはないものとする．

2　関連研究

ふりがなプログラミング [1] はソースコードにふりがなを振ることで理解支援を行うという手法である．ふりがなを意味の通じる文に直した読み下し文も提案されている．ソースコードを読み下し文にした例を図 1 に示す．この手法は，ソースコード上の個々の単語や記号をふりがなとして翻訳しており，字句や文法を理解していない学習者に対して有効である．しかし，分岐や繰り返しの制御文を中心とした一連の計算に適用した場合，各字句を逐一翻訳するので，理解しづらい．つまり，「配列の要素の最大値を求める．」というような処理の理解に対して支援できていない．

```
変数maxに配列tensu[0]を代入する.
max = tensu[0];

変数iに1を代入し，継続条件「変数iが5より下の間」が真の間，以下を繰り返す.
for ( i = 1 ; i < 5 ; i++ ) {
        もし「変数maxは配列tensu[i]より下」が真なら以下を実行する.
        変数maxに配列tensu[i]を代入する.
        if( max < tensu[i] ) max = tensu[ i ];
}
```

図 1　配列の要素の最大値を求めるソースコードを読み下し文にした例

小田らは，統計的機械翻訳を用いた自動コメント生成ツールを提案している [3]．入力言語の構文木と出力言語の単語列が一文ごとに対応した対訳コーパスを用意し，Tree-to-String 翻訳器に学習させることで，ソースコードの一文ごとに対して，最も相応しい単語列を推定する．評価結果からは，7 割以上の文に対して文法的に正しく流暢なコメントを生成した．しかし，複数の文にまたがった一連の計算や，誤ったソースコードに対しての翻訳は想定していない．

Sonia らは，巨大なソフトウェアシステムを対象としたソースコードの要約生成ツールを提案している [4]．大規模なソースコードの重要度が高い箇所を判定し，テキスト検索技法を用いてソースコードから抜き出された単語と構造情報を元に，その箇所の要約を出力することで理解支援を行う．つまり対象ソースコードが大規模であることに対して，要約は一部分である．

3 学習に適した要約の提案

3.1 提案の概要

学習用の典型的な問題に対して，学習者が記述する分岐や繰り返しの制御文を含んだ C 言語ソースコードにおける学習に適した要約の提案を行う．2 章で示した通り，ふりがなプログラミング [1] 等の関連研究では分岐や繰り返しの制御文を含んだ C 言語ソースコードの処理を理解しづらい．学習者にソースコードの処理の理解を促すには，Listing 1 のように複数の文にまたがった一連の計算を簡潔に表現する必要がある．また，学習者は期待した実行結果を得られない誤りを含んだソースコードを記述する場合がある．学習に適した要約として要約から学習者に誤りの発見を促せるものが望ましく，そのため記述したソースコードに誤りが含まれている場合，誤りに気付くことができる表現を用いる必要がある．

Listing 1 配列の要素の最大値を求めるソースコード

```
/* 配列tensu [0]~[4] を比べて，最大値を変数
 max へ代入する．*/
max = tensu[0];
for(i=1;i<5;i++){
   if( tensu[i] > max ) max = tensu[i];
}
```

以上より，学習に適した要約は 2 つの性質をもつ．

- 制御文を中心とした一連の処理を簡潔に表しており，学習者に処理の理解を促す．
- 誤りを含む場合，誤りに気付けるよう表しており，学習者に誤りの発見を促す．

3.2 誤りの分類

誤りに対応した要約を生成するために，繰り返しの制御文を中心としたソースコードに対して学習者が犯しやすいと想定される誤りを整理し，4 種類に分類した．

- 本体の誤り
  制御文中の文 (以下，本体と呼ぶ) において計算を表す記述での誤り．
  例．sum += a[i]; → sum -= a[i];
- 初期値の誤り
  制御文外の文において初期値を表す記述での誤り．
  例．sum=0; → sum=1
- ループ回数の誤り
  制御式においてループの回数と範囲を表す記述での誤り．
  例．for (i=0; i<5; i++) → for (i=1; i<6; i++)
- 条件の誤り
  制御式において条件を表す記述が常に真または偽である誤り．ただし，for(;;) のような意図的と思われるものは除く．
  例．for (i=0; i<5; i–)，for (i=1; i>5; i++)

3.3 誤りの組み合わせ

前節で分類した誤りの組み合わせを表 1 に示した．

表 1 誤りの組み合わせ

	本体	初期値	ループ回数	条件
a.	○	○	○	○
b.	×	○	○	○
c.	○	×	○	○
d.	○	○	×	○
e.	×	×	○	○
f.	×	○	×	○
g.	○	×	×	○
h.	×	×	×	○
i.	-	-	-	×

表 1 において，それぞれの記述が誤っていない場合○，誤っている場合×とした．条件を表す記述が誤っている場合，他の記述の誤りの有無に関係なくソースコードの動作が決まってしまうので - とした．

3.4 誤りに対応した要約例と意図

前節で挙げた誤りの組み合わせ (表 1) に対応して，誤りに対応した要約例と意図を次に示す．しかし f.g.h. については”ループ回数の誤り”の有無以外，他の例と変わらないので省略する．

a. 本体○，初期値○，ループ回数○，条件○ (Listing 2)

Listing 2 配列の合計を求めるソースコード 1

```
/* 配列array [0]~[4] の合計を求める．*/
sum = 0;
for(i=0; i<5; i++){
        sum += array[i];
}
```

一連の計算を簡潔に表した要約を出力する．学習者に対して自身が記述したソースコードの内容を再確認させ，ソースコードの処理の理解を促す．

b. 本体×，初期値○，ループ回数○，条件○ (Listing 3)

Listing 3 配列の合計を求めるソースコード 2

```
/* 配列array [0]~[4] の要素を変数
 sum から引く．*/
sun = 0;
for(i=0; i<5; i++){
        sum -= array[i];
}
```

期待通りでない処理を示した要約を出力する．制御文の本体が期待通りではないことに気づかせる．

c. 本体○，初期値×，ループ回数○，条件○ (Listing 4)

Listing 4 配列の合計を求めるソースコード 3

```
/* 配列array [0]~[4] の合計を初期値
 1 の変数sum に足す．*/
sun = 1;
for(i=0; i<5; i++){
        sum += array[i];
}
```

変数名と初期値の具体的な情報を含ませた要約を出力する．制御文外の文において，初期値が期待通りではないことに気づかせる．

d. 本体○，初期値○，ループ回数×，条件○ (Listing 5)

Listing 5 配列の合計を求めるソースコード 4

```
/* 配列array [1]~[5] の合計を求める．*/
sun = 0;
for(i=1; i<6; i++){
        sum += array[i];
}
```

ループ回数の具体的な情報を含ませた要約を出力する．制御式においてループ回数が期待通りではないことに気づかせる．ループ回数以外は Listing 2 と変わらない．

e. 本体×，初期値×，ループ回数○，条件○ (Listing 6)

Listing 6 配列の合計を求めるソースコード 5

```
/* 配列array [0]~[4] の要素を初期値
 1 の変数sum から引く．*/
sun = 1;
for(i=0; i<5; i++){
        sum -= array[i];
}
```

期待通りでない処理を示し，変数名と初期値の具体的な情報を含ませた要約を出力する．制御文の本体が期待通りではないことに気づかせる．また制御文外の文において，初期値が期待通りではないことに気づかせる．

i. 本体-，初期値-，ループ回数-，条件× (Listing 7)

Listing 7 配列の合計を求めるソースコード 6

```
/* 制御式内の条件が常に真または偽です．*/
sun = 0;
for(i=0; 5<i; i++){
        sum += array[i];
}
```

制御式において，条件が期待通りでなく常に真または偽であることを示す．

4 要約生成ツールの設計・実現

4.1 設計

本ツールは，C 言語ソースコードから 3 章で提案した要約を生成し，ソースコード上にコメントアウトされた要約を挿入する．3 章の提案に基づく要約パターンを用意し，TEBA[2] を用いたパターンマッチングにより，要約対象のソースコードを判別し，必要なソースコード情報の抽出と要約の生成を行う．TEBA の提供するパターンマッチングにおいて，要約を生成する際の問題点を次に示す．

1. ソースコードには異なる記述で同義の処理を行う場合があり，それらのソースコード差異を吸収できない．
2. 提案する要約を実現するために必要な要約ルールの数が膨大になる．

問題点 1 は，i++ のインクリメントを用いた表現を i=i+1 の式に統一する等，学習者が記述する同意義表現を整理して，正規化ルールを定め，前処理にてソースコードの正規化を行うことで学習者の記述するソースコード差を吸収する．問題点 2 は，パターンの書き換え後を示す記述で Perl の手続きを用いることで，書換え前のソースコード情報に応じた要約生成処理を記述できる．また，繰り返し現れる要約生成処理をサブルーチンを用いてまとめることで，再利用性を高め，用意する要約ルールの数を減少させる．

4.2 実現

実装したツールの処理の流れを図 2 に示す．本ツールの処理は，大きく 3 つの手順によって構成される．

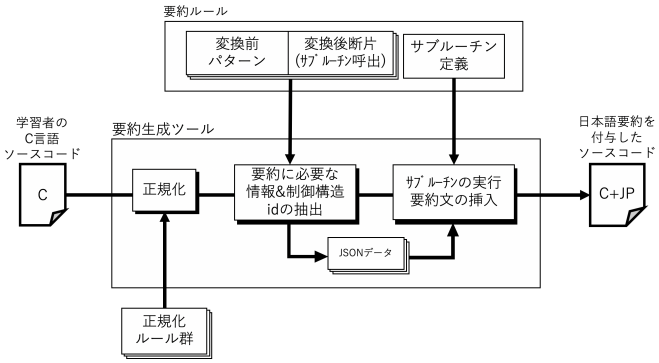

図 2 要約生成処理の流れ

1 つ目は，ソースコードの正規化処理である．正規化ルールに従い，TEBA のプログラム変換器にてソースコードを書換える．ユーザは任意の正規化ルールを拡張できる．2 つ目は，対象ソースコードの判定，必要なソースコード情報の抽出である．これは，要約パターンと要約生成処理を行うサブルーチンを要約ルールとして定義し，TEBA を用いたパターンマッチングで要約対象ソースコードの判定と情報の抽出を行なった．抽出したソースコード情報は JSON 形式でファイルに出力する．なお，ユーザは任意の要約ルールを拡張できる．3 つ目は，抽出情報を元にした

要約の生成である．要約ルールのサブルーチン定義と抽出情報を用いて要約生成処理を行う．得られた要約をソースコード上の該当制御文にコメントとして要約を挿入する．実際に定義したサブルーチンを例に挙げて説明する．

initialize(整数値 1，整数値 2，文字列)

これは変数の初期化を示す要約生成処理を行う．引数として，2 種類の整数値と文字列を受け取る．第 1 引数にソースコード上の初期値を渡し，第 2 引数に基準となる値を渡す．処理として整数値 1 と整数値 2 の比較を行い，真であれば要約を生成せず，偽であれば「初期値 (整数値 1) の変数 (文字列)」という要約を生成する．各プログラムは Perl によって実装し，shell スクリプトでまとめた．

5 評価・考察

5.1 評価

代入や分岐，繰り返し等の制御文を含むソースコードを対象にツールを使用し，期待通りの要約が得られるか検証した．正しいプログラムとして新・明解 C 言語 入門編 [5] のうち制御文を用いた配列の例題 6 題を対象に，誤りが含まれる状況を 16 のテストケースとして分類した．それらに本ツールを適用したところ，すべてのテストケースで 3.3 節で挙げた誤りの種類に応じて要約を生成することができた．

5.2 考察

5.2.1 問題点

TEBA の提供するパターン変換記述では適切に記述できない場合がある．例えば，変数の初期化を含むソースコードにおいて，初期化の式を記述していない場合を考える．学習に適した要約としては Listing 8 のように出力するのが望ましい．これに対応するパターン変換記述を Listing 9 に示す．

Listing 8 配列の合計を求めるプログラム

```
/* 配列 array [0]~[5] を初期値不明の変数
    sum に求める*/
for(i=0;i<5;i++){
        sum+=array[i];
}
```

Listing 9 パターン変換記述

```
%before
for(${init:VAR}=${start:EXPR};${goal:EXPR};${init
    }=${init}+${in:LIN}){
        $[: ${:STMT} $| ${:DECL} $]*
        ${sum:VAR}+=${array:VAR}[${init}];
        $[: ${:STMT} $| ${:DECL} $]*
}
%after
arrayLoop(${start},${goal},${array},${in});
initialize(NULL,0,${array});
sum(NULL, 0);
%end
```

Listing 9 は初期化の式を含んだ正しいパターン変換記述から初期化の式を削除した記述になる．これを要約ルールとし，初期化の式を含んだ正しいソースコードに本ツールを適用すると，初期値を含むパターンとそうでないパターンに同時に適合し，要約が重複してしまう．これを解決するには，ソースコード断片の否定を条件として記述できるよう拡張する必要である．他にも，要約ルールのマッチングの優先順位を設定することが挙げられる．

5.2.2 ツールの拡張

本研究で作成したツールは，要約ルールを追加することで要約対象のソースコードを拡充でき，対応する処理の誤りの種類を増やすことができる．本ツールでは，要約ルールの定義に TEBA のプログラムパターン変換記述と Perl のサブルーチンを用いた．プログラムパターン変換記述は TEBA のプログラムパターン記法に則ってマッチング対象のソースコードを拡張できる．サブルーチンは指定のライブラリに追記することでツールの要約生成処理を拡張することができる．

6 おわりに

本研究では，学習者にソースコードの理解支援のために，学習に適した要約生成ツールを提案した．代入や分岐，繰り返しの制御文に注目して要約を生成することで，プログラムが何を計算するか理解させる．必要な要約ルールを削減するために，要約生成処理をサブルーチンとして部品化，再利用性を高めた．

今後の課題として，パターン変換記述を拡張し，否定を条件とした記述を拡張すること，パターン変換記述の優先順位を設定することや，本研究で対象とした配列，文字列以外の学習単元への適用が挙げられる．

参考文献

[1] リブロワークス：“ スラスラ読める JavaScript ふりがなプログラミング ”，インプレス（2018）．

[2] 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プログラム書換え支援環境 ”，情報処理学会論文誌.53 No.7，pp.1832-1849（2012）．

[3] 小田 悠介，グラム ニュービッグ：“ 統計的機械翻訳を用いた自動コメント生成 ”，ウインターワークショップ 2015・イン・宜野湾 論文集，pp.15-16 (2015)．

[4] Sonia, B., Jairo, A., et al.: Support Program Comprehension with Source Code Summarization (2010)．

[5] 柴田望洋：“ 新・明解 C 言語 入門編 ”，SB クリエイティブ（2014）．