C 言語初学者向けツール C-Helper の現状と展望

内田公太[†]　権藤克彦[††]

C 言語は今でも幅広く用いられ，入門用の言語としてもよく使われる．しかし，C 言語によりプログラミングに入門する初学者にとって，コンパイラが出力するエラーメッセージは理解しにくい．本論文では C 言語初学者向けの静的解析ツール C-Helper の現状と，これからの展望を述べる．C-Helper は初学者が起こしがちなミスを検出し，分かりやすいメッセージで指摘し，解決策を提案する．C 言語としては正しいが，初学者が誤用しやすい機能を把握し，警告する特徴もある．

Conditions and the outlook for C-Helper, a tool for C novices

Kota Uchida[†] and Katsuhiko Gondow[††]

1. はじめに

C 言語は古い言語であるが現在でも幅広く用いられており，大学等でのプログラミング入門用の言語としてもよく使われる．例えば東京工業大学の情報工学科では，1 年次の後期から 2 年次の前期にかけて C 言語を学習する．また「C 言語何でも質問掲示板」[1] はプログラミング初学者からの質問で賑わっている．

第一著者がこの掲示板で活動してきた経験では，C 言語を学ぶプログラミング初学者（以下，特に断らない限り「初学者」は C 言語によりプログラミングに入門する人々のことを指す）は学習に Microsoft Visual C++や GCC を使うことが多い．

これらのコンパイラは，ソースコード上の様々な問題点に対して警告メッセージを出せるため，一見すると初学者に役立つツールである．しかし，出力されるメッセージはある程度プログラミングの経験がないと理解が難しかったり，解決策が書かれていなかったりと，プログラミング初学者向けとは言い難い．

本論文では，C 言語の初学者向け静的解析ツール C-Helper を提案する．C-Helper は C 言語のソースコードを静的に解析し，ソースコードの問題点の指摘や解決策の提案をする．表示されるメッセージをなるべく初学者にとって分かりやすくすることで，プログラミング学習を支援する．

本論文の構成は以下の通りである．まず，2 節でプログラミングの学習がなぜ難しいかを述べ，次に，3 節でプログラミング初学者がよく使う既存のツールとその問題点を指摘する．C-Helper の現状の実装と機能を，実例を用いて 4 節で紹介し，今後追加する予定の機能を 5 節で紹介する．6 節で関連研究について述べ，7 節でまとめる．

2. プログラミング学習の難しさ

C 言語に限らず，プログラミングを効率的に学習するためには，ミスに対する素早く高品質なフィードバックが必要である．学校でプログラミングの授業を受ける場合など，すぐに質問できる人が近くにいる環境では精度が良く情報量も多く，そして素早いフィードバックを期待できるが，入門書などを用いて独習する場合はそうではない．

近くに質問できる人が居ない場合には，プログラミングに関するインターネット掲示板へ質問を投稿するという選択肢がある．そのような掲示板の 1 つに「C 言語何でも質問掲示板」[1] がある．この掲示板では連日のように，C 言語を学ぶ人々から質問や回答が寄せられている．

掲示板への質問には，人に直接質問するのに比べ，質問内容を的確に伝え辛い，回答のレスポンスが遅いなどの欠点がある．質問内容は工夫次第で正確に伝えることはできるが，回答者がインターネットの向こう側のボランティアである以上，レスポンスの改善は難

† 東京工業大学 情報理工学研究科 計算工学専攻
Department of Computer Science, Tokyo Institute of Technology
†† 東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology

しい．

プログラミング初学者は往々にして，最初の投稿で質問内容を正確に伝えることに失敗し，質問者からの質問と，足りない部分を問いただす回答者側からの質問を何度か繰り返すことになる．その結果，回答がさらに遅くなる．プログラミングの学習には試行錯誤が重要であるため，レスポンスの遅さは学習の効率悪化を招く原因となる．

そこで重要になるのが処理系や実行環境からの支援である．問題に出くわした際，最初に得られるのがこれらソフトウェアからのフィードバックだからだ．コンパイラが出力するエラーメッセージや警告メッセージ，実行時のログなどが該当する．特にコンパイラのメッセージには通常，ファイル名と行番号が含まれており，ソースコードのどの部分が悪いか分かるようになっている．

第一筆者が C 言語何でも質問掲示板で活動してきた経験では，C 言語によるプログラミング初学者はコンパイラとして Microsoft Visual C++や GCC を用いることが多い．これらのツールはソースコードの静的解析機能を持ち，ソースコードに潜む問題点を発見できるため，プログラミングの経験者はもちろんのこと，一見すると初学者にも役立つ．

しかし，彼らの質問の中にはコンパイラの出力メッセージを無視していると思われるものが多く存在する．彼らも無視したくて無視しているのではないだろう．3 節で見るように，出力メッセージは初学者には難しい場合があり，また，コンパイラは問題点のある場所は指摘するが，問題点の解決策は提案しない．コンパイラの警告が分かりにくく，また警告に対する具体的な解決策が提案されないため，結局初学者は，出力された警告を活用することなく，掲示板へ質問してしまう．

3. 既存ツールの問題点

2 節で見たように，プログラミング学習には処理系や実行環境の支援が重要である．独習する場合であっても，誰かから教わる場合であっても，これらのソフトウェアの出力メッセージを最初に読むことになる．

しかし，出力されるメッセージは，ある程度プログラミング経験がないと理解が難しいことが良くある．加えて，どう直せばいいのかという情報は出力されないことが多い．例えば GCC は次のコードに対し “comparison between pointer and integer” なる警告を出す．

```
int word = '+';
if ("+" == word) {}
```

この警告は，ある程度経験がある C 言語プログラマが読めば理解できるのだが，初学者には通じない場合がある．少なくとも，ポインタを学習していない段階では “pointer” という用語を理解できない．この場合，対処法の 1 つは `"+"` を `'+'` に変更することだが，GCC はいかなる対処方法も提案してくれない．

さらに，Microsoft Visual C++や GCC は，C 言語としては正しいが初学者の意図に反するようなプログラムに対して，当然かもしれないが何も警告しない．

例えば GCC は次のコードに対して何の警告も出さない（環境によっては `printf()` のパラメタ不整合に関する警告は出る可能性がある）．なぜなら C 言語としては正しいコードだからだ．しかし，このコードの作者はポインタ変数の大きさではなく，動的に確保した配列の要素数 `100` を表示させたかったのである．

```
char *p = malloc(100);
printf("%u\n", sizeof(p));
```

Microsoft Visual C++や GCC はコンパイラであり，解析は主機能でない．ソースコードを検査するためのツールとして Splint[2] がある．

Splint はソースコードのセキュリティ欠陥やその他の間違いなどを静的に検査し，様々な問題を発見できる．例えば，C-Helper（後述）が検出する「`return` 文の不足」は Splint でも検出可能である．

しかし，Splint は Microsoft Visual C++や GCC と同様に，問題点を指摘するだけであり，解決策は提案しない．`char` 型配列の 1 つの要素へ文字列リテラルを代入しようとしたときも，型が違うというメッセージしか出さない．

関連したツールに CX-Checker[3] がある．6 節で詳しく述べるが，CX-Checker はユーザが定義したルールに対し，与えられたソースコードが合致しているかを検査するソースコードチェッカである．CX-Checker はカスタマイズ性が高く，様々な問題を検出するルールを簡単に書ける．

しかし私が調査した範囲では，CX-Checker は `printf()` のパラメタ間違いを検出できない，ポインタ解析が必要となる問題の検出も難しい，解決策の提案機能がない，などの点で C-Helper とは異なる．

4. C-Helper

C-Helper は著者らが開発している，C 言語初学者向けの静的解析ツールである．入力されたソースコードの字句情報，抽象構文木，制御フローグラフ，到達

定義解析，ヒューリスティクスを組み合わせ，初学者が犯しがちな間違いを静的に検出する．検出した問題点は，初学者に分かりやすい言葉を使い，解決策も交えて提示する．

開発の動機は，3 節に示したように，初学者に対し，既存のコンパイラより分かりやすい表現でフィードバックを返したい，C 言語としては間違っていないが，初学者が意図しない使い方をしがちな機能を使った場合に警告したい，という 2 点である．

C-Helper の実装に用いている抽象構文木や制御フローグラフの生成，到達定義解析の技術には既存の技術を使っており，新規性はない．ヒューリスティクスの部分は，C 言語何でも質問掲示板の調査で得た知見を生かしている．C-Helper は，既存の技術に少しのヒューリスティクスを組み合わせ，どれほどの種類の間違いを発見し，どれほど初学者に分かりやすいメッセージを出力できるかを確認するために実装している．

本論文で想定している「初学者」は，プログラミング自体に入門しようとする人である．例えば大学のプログラミングの授業を初めて受ける学生，入門書を買って間もない人などを想定している．

4.1 検出できる問題

C-Helper は本論文執筆時点で以下の問題を検出できる．それぞれの問題の典型的なコード例と，C-Helper が出力する警告メッセージを紹介する．掲載したコード例は C 言語何でも質問掲示板[1] に投稿された実際のコードを，問題の本質を変えない程度に簡略化したものである．

(1) インデント乱れ，
(2) `char` 型変数への文字配列の代入，
(3) `printf()` のパラメタ不整合，
(4) `return` 文の不足，
(5) 関数定義の余分なセミコロン，
(6) 構造体宣言のセミコロン不足，
(7) 動的に確保した配列に対する `sizeof` の使用，

図 1 インデント乱れ（タブとスペースの混在）

```
void f(void) {
    int i;
␣␣␣␣int j;
}
```

警告メッセージ「インデントに用いる文字は統一すべきです．前方ではスペースが，ここではタブが用いられています．」

初学者のコードを見ていると，インデントがまったく揃っていない場合がある．インデントの乱れ方には

図 2 インデント乱れ（インデント幅の乱れ）

```
void f(void) {
int i;
    while(!foo()){
for(i=0;i<20;i++){
  bar();}
    }
}
```

警告メッセージ

- test.c:2:1:行頭から書き始めるのは分かりにくいため、スペース 4 個分インデントすべきです。
- test.c:4:1:インデントが乱れています。スペース 8 個分インデントすべきです。
- test.c:5:1:インデントが乱れています。スペース 12 個分インデントすべきです。

a) スペースとタブが混在している（図 **1**），b) インデントの深さがそろっていない（図 **2**），c) そもそもインデントされていない，の 3 種類があり，C-Helper はそれらに対応している．

インデントの乱れを警告しようとすると，基準とするインデント規則を決める必要がある．しかし，インデントの好みは人によって異なり，絶対的な規則はない．企業では統一的な規則を定める場合もあるが，初学者がプログラミングを学ぶ段階では固定した規則は存在しない．一番最初に読んだ本，受講した講義などに影響を受けることもあるだろう．

インデント規則は特に正解があるわけではなく，プロジェクトの中で一貫していればよい．そこで C-Helper は，CX-Checker[3] のように固定したルールに照らし合わせる方法ではなく，最初に検出したインデントを基準とし，その後のインデントを検査する．こうすることで，初学者の意思を尊重しつつ，インデントの乱れを防げる．

図 3 `char` 型変数への文字配列の代入

```
void f(void) {
    char arr[20];
    arr[20] = "foo";
}
```

警告メッセージ「char 型配列の 1 つの要素に文字列を格納できません．strcpy を使うことを検討してください．」

図 **3** に示す問題は，`char` 型配列の初期化とその他の場合での文字列データの格納方法が違うことに起因する．C 言語では，配列の定義時に初期値として`"foo"` のような文字列リテラルを指定できるが，定義時以外で文字列データを格納するには，配列の各要素に別々

に代入するか，strcpy() などを使う必要がある．

図 3 のコードを見ると，配列の定義時以外で文字列リテラルをそのまま代入しようとしている．恐らくこの人は代入式の左辺値を char 型だと認識しておらず，「arr[20] と書くと文字配列全体を表す」と思っているのだろう．なぜなら，文字配列の定義時ではそうだからである．

このコードを GCC と Splint に入力すると，以下のメッセージを得る．

- GCC
  warning: assignment makes integer from pointer without a cast
- Splint
  Assignment of char * to char: arr[20] = "foo"
  Types are incompatible.

GCC より Splint のメッセージの方が詳しいが，どちらも C-Helper とは違い，問題の解決策は提案しない．さらにどちらのメッセージも，C 言語のポインタを用いて問題を説明している．C 言語の入門書では，ポインタを学習する前に文字列の表示や配列を学習することが多いため，その段階の初学者に対して，ポインタという用語を使うと混乱を招く可能性がある．

図 4 printf() のパラメタ不整合

```
#include <stdio.h>
void f(void) {
    printf("%d", 3.14);
}
```

警告メッセージ「引数と%変換の型が合いません. (浮動小数点数の表示には %f, %e, %g などが使えます.)」

図 4 は，初学者が良く犯すミスの 1 つである，printf() のパラメタ間違いの様子を示す．このミスは C-Helper に限らず，GCC や Splint でも検出できた．GCC の出力メッセージは次である．"warning: format ‘%d’ expects type ‘int’, but argument 2 has type ‘double’"

Splint もほぼ同様なメッセージであった．どちらとも，%d は int 型を期待しているが，渡されたのは double 型である旨を報告している．しかし，具体的にどの書式指定子を用いれば良いのかという提案はなかった．C-Helper は使える書式指定子の候補を示すことにより，初学者の学習を効率化する．

C-Helper は，戻り値の型が void 以外であるにもかかわらず，値付きの return を実行しないパスが存在する関数を検出する（図 5）．ただ，現在はまだ分岐の条件式を考慮しないため，明らかに通らない分岐があっても通る可能性があるとして解析する．

図 5 return 文の不足

```
int f(int i) {
    if (i < 0) {
        return -i;
    }
}
```

警告メッセージ「return 文がありません.」

GCC と Splint，Visual C++ 2010 もこの問題を検出する．GCC と Visual C++ 2010 は条件式の値を考慮しており，図 5 の if 文の条件式を!0 に変更すると警告は出なくなる．

図 6 関数定義の余分なセミコロン

```
int main(void);
{
    // ...
}
```

警告メッセージ「関数定義にはセミコロン ; を付けません.」

図 6 は，関数を定義しようとしたのだが，間違えてセミコロンを付け足してしまい，意図せず関数プロトタイプ宣言になってしまうというパターンである．このソースコードを Splint で解析しようとしたところ，構文エラーとなり，関数定義のつもりがプロトタイプ宣言になってしまった，という意図は検出できなかった．

また，GCC は“error: expected identifier or ’(’ before ’{’ token” というメッセージを出した．このメッセージは初学者を混乱させるだろう．本当は余計なセミコロンを消さねばならないのに，“expected” という単語から，何か書き足りないことがあるのではないか，と思ってしまう恐れがある．

図 7 構造体宣言のセミコロン不足

```
struct st {
    int i;
}
```

警告メッセージ「構造体の宣言にはセミコロンが必要です.」

図 7 は図 7 とは逆に，セミコロンが必要な場所でセミコロンを付け忘れたパターンである．C 言語では，構造体宣言にはセミコロンが必要なのだが，それを書

き忘れてエラーとなっている．Splint も GCC も，構文エラーを知らせるメッセージを出すだけであった．

図 8　関数の戻り値で構造体を宣言

```
struct st { int i; } f(void) {
    /* do something */
}
```

警告メッセージ「構造体の宣言にはセミコロンが必要です.」

もし，構造体宣言が関数の戻り値型として使われている場合（図 **8**），構造体宣言の閉じ括弧}の後ろにセミコロンを書いてはならない．この場合，C-Helper は警告を出さない．

C-Helper はセミコロンがない場合に加え，セミコロンが遠くにある場合も指摘を行う．例えば構造体宣言の閉じ括弧}の 2 行下の行にセミコロンを書いた場合，次のメッセージを出力する．「構造体の宣言のセミコロンは，最後の閉じ括弧}の直後に書くと見やすくなります.」

C 言語は空白自由言語であるから，空白をいくら入れようと，コンパイルや実行にはまったく影響がないが，過度な空白はソースコードの可読性を落とす原因となる．C-Helper は「おかしな」空白の使い方を指摘することで，初学者が可読性の高いソースコードを書けるよう支援する．

図 9　動的に確保した配列に対する `sizeof` の使用

```
#include <stdlib.h>
#include <stdio.h>
void f(void) {
    char *p = malloc(128);
    printf("%d\n", (int) sizeof(p));
}
```

警告メッセージ「sizeof(p) は 128 ではなく 4 を返します（仮定 6）．それは本当に意図したことですか？」
情報メッセージ「仮定 6: ポインタ変数のサイズを 4 バイトと仮定しています.」

図 **9** に示した例は C-Helper の特徴を端的に表している．C 言語の規格では，ポインタ変数に `sizeof` を適用することは，ポインタ変数の大きさを取得する正しいやり方である．したがって，通常のコンパイラはこれを警告またはエラーとして検出しない．

しかし，ポインタ変数が指す先の配列の大きさを取得しようとして，初学者はこの操作をする場合がある．もちろんこれは間違いである．そこで C-Helper はその操作が本当に意図したことなのかと問いかけ，初学者がミスに気づく機会を与える．

ところで，ポインタ変数の大きさは処理系に依存しており，`sizeof(p)` が 4 を返すという説明は不正確である．そこで，C-Helper は処理系に依存した仮定を用いた場合に，その仮定を明示することにした．それが図中の「情報メッセージ」である．

4.2　C-Helper の実装

ここでは C-Helper の実装について述べる．C-Helper は Eclipse[4] プラグインとして，Eclipse CDT[5] の解析機能を用いて実装している（図 **10**，図 **11**）．

Eclipse のプラグインとして開発することで，開発環境としての基本機能は Eclipse に頼り，本来行いたいソースコードの解析機能の実装に力を注げる．また，Eclipse CDT のパーサは構文エラーがある場合もできるだけ解析を続け，それらしい抽象構文木を出力できるため，構文エラーが多いと予想される初学者のソースコードの扱いに適していると考えられる．

ソースコードを解析し，問題や解決策を発見した場合，警告マーカを生成することで，Eclipse の Problem Views への表示と，ソースコードの該当箇所のマーキングを行う．

C-Helper がソースコードを解析する手順は以下の通りである．

- Eclipse CDT のパーサを用いて，入力されたソースコードに対応する抽象構文木を生成する．
  この段階で，インデント乱れ，`char` 型変数への文字配列の代入，`printf()` 関数のパラメタ不整合，関数定義の余計なセミコロン，構造体宣言のセミコロン不足，を検出できるようになる．
- 抽象構文木を用いて制御フローグラフを生成する．
  この段階で，`return` 文の不足を検出できるようになる．
- 制御フローグラフを用いて到達定義解析を行う．
  この段階で，動的に確保した配列に対する `sizeof` の使用を検出できるようになる．

5.　C-Helper の展望

これからやるべきことが 2 つある．C-Helper の機能拡張と評価実験である．

C-Helepr の機能はまだ少なく，初学者が間違いやすいパターンの検出機能をさらに拡充する必要がある．以下，現段階で実装を考えている機能について述べる．

5.1，5.2，5.3，5.4 で述べる解析は，我々が知る限

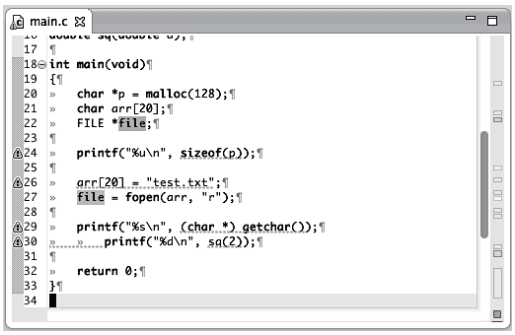

図 10　ソースコード編集画面

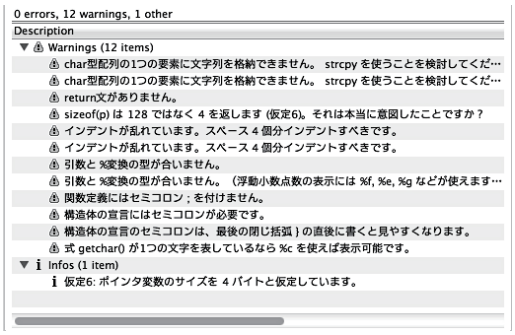

図 11　警告一覧

り，一般的な状況において正確に行うことは難しいものだ．解析対象を初学者のソースコードと限定することで，役立つ解析ができると考える．5.5 で述べる機能は，実現は難しくないと考えるが，実装の都合で未搭載である．

5.1　取りうる値の範囲に基づく検査

図 12　ナル文字を忘れてバッファオーバーラン

```
char buf[2];
int i = rand() % 100;
sprintf(buf, "%d", i);
```

整数型の変数の値が取りうる範囲を解析し，その情報に基づいてソースコードの問題を探す．例えば，典型的な for 文の使い方において，カウンタ変数が取りうる値を解析すれば，配列のバッファオーバーランを発見したり，更新式を間違えたことにおける無限ループなどを検出可能である．

図 **12** で示すコードは，0 から 99 の乱数を生成し，文字列へ変換するプログラムである．乱数は最大でも 2 桁であるから，バッファは 2 文字分で良いと判断したのだが，実はナル文字を格納するために追加で 1 文字分必要である．

この問題を発見するには，変数 `i` が `sprintf()` 実行時に取りうる値の範囲を知る必要がある．その計算には抽象領域を用いた抽象解釈を用いることができる．さらに，`sprintf()` に関する仕様を組み込んでおく必要もある．

5.2　標準ライブラリの仕様に基づく検査

C 言語の標準ライブラリに含まれる関数の仕様を C-Helper に組み込んでおき，その仕様に違反するソースコードを検出する．

例えば，`fread()` の第 2，3 引数の読み込みバッファのサイズに関する仕様を組み込んでおけば，図 **13** の検出が可能になるだろう．

図 13　関数内部でバッファオーバーランが起こる

```
FILE* fp = fopen("foo", "r");
char buf[100];
fread(buf, 1, 10000, fp);
```

また，`fopen()` の第 2 引数のファイルオープンモードに関する仕様と `fread()` は読み込みモードで開いたファイルポインタを要求するという仕様を組み込んでおけば，図 **14** の検出が可能になるだろう．

図 14　オープンモードの不整合

```
FILE *fp = fopen("foo", "w");
char buf[1024];
fread(buf, 1, sizeof(buf), fp);
```

書き込みモードでファイルを開いているにもかかわらず，読み込もうとするのは誤り．

5.3　リソース解放忘れの検査

ヒープメモリやファイルなどのリソース解放忘れを，ポインタ解析を行い，精度よく検査したい．例えば図 **15** におけるファイルポインタ `fp` に対しポインタ解析を行うことで，ファイルの閉じ忘れを検出できる．

5.4　大域的な解析

現在 C-Helper が行える解析の中で，制御フローグラフや到達定義解析に基づくものは，関数単位での解析である．初学者に役立つ解析をさらに行うためには，関数間の解析が必要となる．例えば，ある変数を値渡しで関数へ渡し，その関数の中で対応する仮変数を変

図 15　ファイルポインタの解析

```
FILE *fp;
while (1) {
    fp = fopen("foo", "r");
    /* do something with fp */
}
```

更しても，呼び出し元の変数は変更されない，というメッセージを表示したい．

関数間だけでなく，ファイル間の解析も実装したい．例えば，複数のヘッダファイルに，構造体タグ名だけが異なるような構造体宣言が存在することを検出するには，それらのヘッダファイルにまたがるクローン検出が必要である．

また，ファイル分割の方法を学習したばかりの人は，しばしばヘッダファイルに変数や関数の実体を定義してしまう．それを検出するには，.c ファイルでインクルードされているすべてのヘッダファイルを解析する必要があるだろう．この検出はそれほど難しくないと考えられるが，実装の都合でまだ搭載していない．

5.5　重複した警告を出さない機能

静的解析では一般に，怪しい部分をなるべく多く検出しようとすると，誤検出が多くなる．実際に実行させるわけではないため，これは本質的に仕方のないことである．

しかし，いくら仕方のないことだと言っても，実際には起こらないと分かっている（または意図して起こしている）問題について，ツールを適用するたびに表示してしまうと，ユーザはメッセージを読まなくなってしまう．これでは解析を行う意味がない．

そこで，選択したメッセージについて，再度表示させない機能を実装しようと考えている．例えば Eclipse CDT ではこの機能を持っており，メッセージの種類ごとに存在するチェックボックスのチェックを外すことで，以後の解析ではその種類のメッセージは出力されなくなる．

しかし Eclipse CDT のこの機能は，選択を外した種類の警告がまったく出力されなくなってしまう．C-Helper では，メッセージの種類に加え，メッセージが対象とするソースコード断片の位置情報を利用し，抑止すると決めたソースコードのその部分についてのみ，以後の解析でメッセージを抑止しようと考えている．

これは例えば，あるソースコードでは意図的に変則的なインデントをしており，インデント乱れ警告を抑止したとしても，次に解析するソースコードではインデント乱れを検査したい可能性があるからだ．

6. 関 連 研 究

この節では，本研究に関連する既存の研究を紹介し，本研究と比較する．

6.1　プログラム全体の解析を対象とした研究

Song らは C 言語初学者向けのインテリジェントな指導システム，C-Tutor を開発した[6]．C-Tutor は，入力されたソースコードを，プログラマの意図を使って解析する．プログラマの意図は，教員が用意したサンプルプログラムをリバース・エンジニアリングし，プログラム記述（program description）の形で得る．

その後，知識ベースのプログラム解析器によりバグを探す．この解析器は動的解析と静的解析を両方行う．動的解析により，プログラム記述のどの目標が満たされて，どれが満たされていないかを調べ，次のパターンマッチの高精度化に役立てる．

最後に，プログラム記述にある目標のうち 1 つを選択し，その目標を実装するプランの集合を知識ベースから取得する．各プランとソースコードを一致させることを試みる．そして，先の動的解析の結果を用いつつ，プランとソースコードの相違を検出し，学習者に報告する．

この手法は，ソースコードがビルド・実行可能でなければならない．さらに，正解となるプログラムが用意されていなければならない．授業などで明確な課題が与えられており，そのために作成しているプログラムには適用可能だが，自分で作りたいと思ったプログラムを作ろうとする場合や，授業であっても，要素技術を確認するための補助プログラムなどを作ろうとする場合には適用が難しい．

Truong らは Java の初学者向け静的解析フレームワークを提案した[7]．Web ベース学習の支援を目的としたもので，専用の Web サーバ上で動作する．このフレームワークは，入力されたソースコードのメトリクスを計測し，模範解答との相同性を解析することができる．

まず，学生が作成したソースコードを ANTLR パーサを用いて XML 表現の抽象構文木へと変換する．次に，得られた抽象構文木を用いてソフトウェア工学メトリクスを計測する．その後，学生が作成したプログラムと模範解答の抽象構文木を変換し，抽象表現を得る．2 つの抽象表現の差異を識別し，学生に追加のフィードバックを送信する．

このフレームワークには標準で，未使用変数の検出，タブが混じっていないことの確認，プログラムの循環的複雑度の測定など，11 の項目を検査できる．フレーム

ワークは設定や拡張が可能であり，StaticAnalysis クラスを継承したクラスを書くことで，独自の検査項目を実装することもできる．

相同性解析を使わず，ソースコードの静的解析だけならば，正解プログラムなしでフレームワークを用いて解析できる．しかし，標準ではデータフロー解析やポインタ解析など，高度な解析は実装されていない．

大須賀らは，カスタマイズ可能なソースコードチェッカ CX-Checker を提案した[3]．CX-Checker は，ユーザが事前に定義したルールに対し，与えられたソースコードが合致しているかを検査する．

まずソースコードが構文解析され，XML 表現の抽象構文木となる．この XML 表現には，構文構造に加え，定義参照関係，データ・制御フロー情報，型情報が含まれる．ユーザは，この XML ファイルから得られる情報を使い，ルールを記述する．XPath を使うことで，かなり表現力を保ちながら，簡単にルールを記述できる．

しかし，XML 表現にはポインタ解析の情報や，printf() の書式文字列の情報などは含まれておらず，高度な解析をしたい場合は，ルールの一部として自分で解析器を実装する必要がある．また，CX-Checker は解決策を提案する機能を持たない．これらの点で，C-Helper とは異なる．

6.2 printf() の解析を対象とした研究

Cowan らは，printf() に不正な書式文字列を渡すことによる攻撃を検出する FormatGuard を提案した[8]．この攻撃手法は 2000 年 6 月に広く知られることになった．printf() のような関数の書式文字列に %n を含ませることにより，メモリの好きなアドレスへ好きなデータを書き込める攻撃手法である．

この攻撃は，ユーザからの入力文字列を，そのまま printf() の第 1 引数として指定することで成立する．printf() の第 1 引数に文字列リテラルを指定すればこの問題は起こらないが，Cowan らによれば，プログラマが“printf("%s", str)”を省略して“printf(str)”と書くと，その問題が起こり得る．

C-Helper は，書式文字列がリテラルの場合のみ，書式文字列の解析を行う．書式文字列が変数の場合，文字列リテラルを指定するように警告する．

7. 結　　論

本論文では，プログラミングの学習を効率良く行うには処理系の支援が必要であることを述べ，初学者の学習を支援する静的解析ツール C-Helper を提案し，実装した．

まず，C 言語は今でも入門用言語としてよく使われることを述べた．大学の授業や C 言語向けの掲示板などの様子から，C 言語はプログラミングの入門用として頻繁に用いられる．

次に，初学者がよく使う既存のコンパイラなどについて，その問題点を指摘した．初学者が良く使う Microsoft Visual C++や GCC は，初学者にとって難解なメッセージにより問題を指摘すること，また，問題の解決策は提案しないことから，初学者にとってはあまり役に立たない場合があることを述べた．

最後に，C-Helper が現状で検出できる問題を紹介し，実装について簡単に説明した．C-Helper は Eclipse のプラグインとして実装してあり，本論文執筆時点では (1) インデント乱れ，(2) char 型変数への文字配列の代入，(3) printf() のパラメタ不整合，(4) return 文の不足，(5) 関数定義の余分なセミコロン，(6) 構造体宣言のセミコロン不足，(7) 動的に確保した配列に対する sizeof の使用，の 7 つの問題を検出できる．

謝辞　C-Helper はサイボウズ・ラボユースの支援のもと製作している．この場を借りて感謝の意を表する．

参　考　文　献

1) C 言語何でも質問掲示板, http://dixq.net/forum/viewforum.php?f=3 (2012).
2) Splint, http://www.splint.org/ (2012).
3) 大須賀俊憲, 小林隆志, 渥美紀寿, 間瀬順一, 山本晋一郎, 鈴村延保, 阿草清滋: CX-Checker:柔軟にカスタマイズ可能な C 言語プログラムのコーディングチェッカ, 情報処理学会論文誌, Vol. 53, No. 2, pp. 590–600 (2012).
4) Eclipse, http://www.eclipse.org/ (2012).
5) Eclipse CDT, http://www.eclipse.org/cdt/ (2012).
6) Song, J. S., Hahn, S. H., Tak, K. Y. and Kim, J. H.: An intelligent tutoring system for introductory C language course, *Comput. Educ.*, Vol. 28, No. 2, pp. 93–102 (1997).
7) Truong, N., Roe, P. and Bancroft, P.: Static analysis of students' Java programs, *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*, ACE '04, Darlinghurst, Australia, Australia, Australian Computer Society, Inc., pp. 317–325 (2004).
8) Cowan, C., Barringer, M., Beattie, S., Kroah-Hartman, G., Frantzen, M. and Lokier, J.: FormatGuard: automatic protection from printf format string vulnerabilities, *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, Berkeley, CA, USA, USENIX Association, pp. 15–15 (2001).