C 言語における実引数が NULL の関数呼出のパターンに関する調査と考察

2018SE004 江坂亜純　2018SE012 原野礼衣
指導教員：蜂巣吉成

1　はじめに

プログラマは他者が書いたソースコードを読み，処理内容を理解して利用や編集を行う．関数の定義箇所を探してその処理を理解する必要や書き方があっているか考えながら編集する必要があり，手間である．特に C 言語では処理を関数にまとめ，関数を呼び出すことで処理が行われるがその理解に時間がかかる．その問題の原因として，次の 2 つがある．

原因 1　関数がどのような処理を行うのか定義を確認する必要がある．

原因 2　通常，関数呼出の引数は関数の実行に必要な情報を与える働きをするが，NULL であった場合確認しなければならないことが多い．

原因 1 について，どのような処理を行っているかを確認したいときには定義を確認する必要がある．しかし，定義箇所を探すのに時間を要する．原因 2 について，NULL は実行時エラーの原因になりうるので扱いに気をつけなくてはならないリテラルである．なぜ NULL になっているのか，本当に NULL で適当か，他の関数呼出のときに NULL が使われているか，NULL のときとそうでないときはどのような違いがあるのかを確認する必要がある．これらを確認するために，関数定義の確認，NULL になっている仮引数の確認とその仮引数の処理，他の呼出部分の確認をするために多くの時間を要する．原因 1 に関する研究は行われているが [3][4]，原因 2 に関する研究は見当たらなかった．

本研究では，開発する際の確認する作業の負担を減らすために，関数呼出で実引数が NULL のものに注目し，NULL についてどのような使われ方をするかユーザに提示する．その提示を行うために，実引数に NULL が使われている関数のソースコードに共通のパターンがあると仮定し調査する．パターン分けを行うために，2 つを調査する．

調査 1　実引数が NULL の関数呼出を調査する．

調査 2　実引数が NULL の関数の定義を調査する．

NULL の使われ方についてパターン分けし，各パターンに対して簡潔な説明をつける．その後，どのパターンに適合するかを自動で判別するツールを作成する．そのツールの有効性を確認するため，他のソースコードで適用実験をする．

2　関連研究

1 節の原因 1 に関して理解支援の研究は行われている．鈴木ら [3] は既存のソースコードの理解を容易にするため可視化によるコードリーディングのツールを提案している．太田ら [4] はクロスリファレンサと視覚化ツールを用いて，C プログラムの理解支援のシステムを提案している．そして，関数呼出経路による動作の変化の理解支援を実装している．原因 2 に注目した理解支援は調べたかぎり見つからなかった．そのため，本研究では，原因 2 の NULL に関する開発支援を行う．

3　実引数が NULL の関数呼出のパターン調査

3.1　調査概略

本研究では，実引数が NULL のものに注目し，パターン分けからなぜ NULL になっているのか理由を提示する．調査の流れを説明する．まず，すべての C ソースファイルから関数ごとに関数呼出として使われている部分と定義部分を抽出する．抽出した関数呼出について，全体の呼出回数，NULL が引数の時の呼出回数，各引数の NULL が使われている回数をカウントした．その結果，関数が可変長引数のものがあったため，引数の最大値-最小値と最後の引数が NULL になる回数もカウントした．調査対象は Apache httpd-2.4.46[1] の C ソースファイルで，必要に応じて Apache Portable Runtime[2] を使用する．

3.2　関数呼出の調査

調査 1 の実引数が NULL の関数呼出を調べる．関数呼出として使われている部分を見ると，図 1 のように関数呼出時に引数の個数が不定かつ最後の引数が NULL になっているものがあった．今回の調査では NULL に注目しやすいよう実引数が NULL でないものは「#」に書き換えてある．

```
apr_pstrcat(#,#,#,#,#,NULL,)
apr_pstrcat(#,#,#,NULL,)
apr_pstrcat(#,#,#,#,#,#,NULL,)
apr_pstrcat(#,#,#,NULL,)
apr_pstrcat(#,#,#,#,NULL,)
apr_pstrcat(#,#,#,#,#,NULL,)
```

図 1　apr_pstrcat の関数呼出

他の関数についても分析し，表にまとめると表 1 のようになる．引数の最大個数-最小個数をすると 0 になっていないことから引数の個数が不定であることが読み取れる．また，呼出回数と最後の引数が一致していることからすべての関数呼出で最後の引数が NULL になっていることが読み取れる．

全体をまとめると引数の個数が不定な関数は 33 あり，そのうち呼出回数と最後の引数が一致し，NULL になっているものは 30 であった．これにあてはまらないものは定

表 1 可変長引数の関数呼出

関数名	最大-最小	割合	呼出回数	最後 NULL
apr_pstrcat	10	100	684	684
ap_rvputs	8	100	112	112
…	…	…	…	…

義が複数ありどの定義を使うかによって引数の個数が異なるというものであった．

これらの調査から，同じ関数でも実引数の数が異なりかつ最後の引数が NULL になるパターンがあることがわかった．これは「可変長引数の最後を表す NULL」といえる．

3.3 関数定義の調査

次に調査 2 の関数定義を調べる．3.1 節で抽出した関数定義の NULL に関わる部分を調査し，特徴から分類した．

3.2 節でみた関数定義をみると，ソースコード 1 のように仮引数の最後が... で終わっている．

ソースコード 1 apr_pstrcat の定義

```
1 apr_pstrcat(apr_pool_t *a,...,)
2 {
3     char *cp, *argp, *res;
4     apr_size_t saved_lengths[
          MAX_SAVED_LENGTHS];
5     int nargs = 0;
```

このことから可変長引数の最後の NULL のパターンの時，関数定義が hoge(a,b,c,...) であるといえる．

ほかの関数についても調査する．関数呼出時に NULL が使われている引数を ptr とする．ptr についての記述を関数定義から探すと if 文の条件式に使われていることが多かった．そのため以下のように分類する．

A if(ptr) または if(ptr!=NULL)

B if(!ptr) または if(ptr==NULL)

C if 文なし

if 文内に ptr があるものは，ポインタの指す先をみている*ptr と ptr->xxx，ポインタ自身をみている ptr という使われ方に分類した．また，ptr の書かれ方として，=の左辺に書かれるものと=の右辺に書かれるもの，関数の引数に使われるものがあった．以上より，ptr の使われ方を分類し表にまとめると表 2，3，4 になる．

表 2 *ptr のパターン分け

	*ptr=	=*ptr	func(…,*ptr,…)
A	1-1	1-2	1-3
B	2-1	2-2	2-3
C	3-1	3-2	3-3

if 文内の分類について，これのどれにも当てはまらないものは例外とする．条件演算子は if 文に直してから分類した．2-1 から 2-6，3-1 から 3-6 は実行時エラーとなるので，実際にありうる記述は 1-1 から 1-9，2-7 から 2-9，3-7 から 3-9 となる．

表 3 ptr->xxx のパターン分け

	ptr->xxx=	=ptr->xxx	func(…, ptr->xxx, …)
A	1-4	1-5	1-6
B	2-4	2-5	2-6
C	3-4	3-5	3-6

表 4 ptr のパターン分け

	ptr=	=ptr	func(…, ptr, …)
A	1-7	1-8	1-9
B	2-7	2-8	2-9
C	3-7	3-8	3-9

NULL になりうる引数の書かれ方がどのパターンに当てはまるか分類し，各パターンの検出数，割合 (%) を求めたものが表 5 である．

表 5 apache の検出結果

パターン番号	検出数	割合 (%)
1-1	10	12.99
1-2	1	1.30
1-3	0	0.00
1-4	0	0.00
1-5	1	1.30
1-6	4	3.90
1-7	1	1.30
1-8	8	10.39
1-9	9	12.99
2-7	9	10.39
2-8	0	0.00
2-9	0	0.00
3-7	0	0.00
3-8	25	25.97
3-9	32	38.96

実行時エラーになりうる 2-1 から 2-6，3-1 から 3-6 は 0% になっていたので，この数値からもエラーになりうる記述だということがわかる．これらの数値が 0% より多い値であれば，記述が間違っている可能性がある．1-3，1-4，1-5，2-8，2-9，3-7 のパターンについて Apache では当てはまる記述が見つからなかったが，Apache に例がなかっただけであり，実行時エラーではない．

実際にあり得る 15 パターンについてパターン定義する．そのパターンについて使われ方をまとめると表 6 となる．

以上の調査から，関数呼出，関数定義で使われ方の特徴を見つけ，パターンを定義し分類した．

表 6　パターンの使われ方

パターン	使われ方
1-1	!NULL のとき，ポインタの指す先に代入
1-2	!NULL のとき，ポインタの指す先を代入
1-3	!NULL のとき，関数の引数のひとつ
1-4	!NULL のとき，構造体メンバに代入
1-5	!NULL のとき，構造体メンバを代入
1-6	!NULL のとき，関数の引数でどこかを指す
1-7	!NULL のとき，ptr に代入
1-8	!NULL のとき，ポインタを代入
1-9	!NULL のとき，関数の引数のひとつ
2-7	NULL のとき，ptr を設定
2-8	NULL のとき，代入
2-9	NULL のとき，関数の引数のひとつ
3-7	ptr を設定
3-8	右辺代入
3-9	関数の引数のひとつ

4　パターン分類ツール

4.1　パターン分類ツールの概要

前節までで手動でパターン分類していたものを自動で分類するツールの作成を行う．分析したいフォルダ名を入力すると，作成されたすべてのファイルが調査するフォルダ内のフォルダ RESULT に出力する．各関数の定義，関数呼出に関する統計，パターン検出結果が出力される．

4.2　全体の流れ

本研究で作成するツールの内部構造は図 2 のようになる．

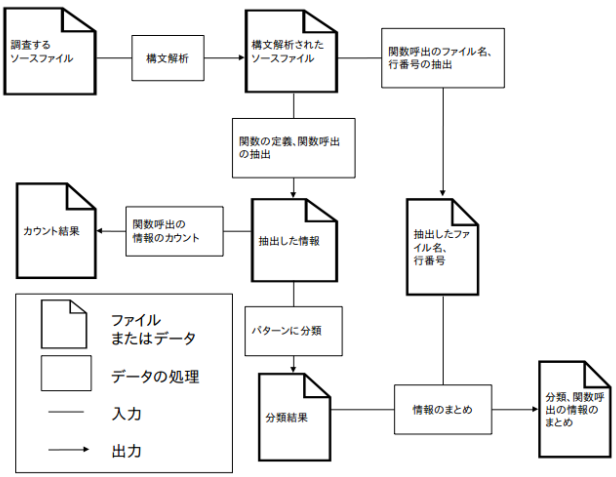

図 2　ツールの内部構造

4.3　パターンマッチング

3 章で実引数の NULL の使われ方のパターンを定義した．3 章では手動で関数がどのパターンに当てはまるか分類していたが，それを自動で分類するツールの作成をする．3.3 節の表の 1-1，1-2，1-4，1-5，1-7，1-8，2-7，2-8，3-7，3-8 を対象に TEBA[5] でパターンを書く．ひとつのパターンでも様々な書き方で使われていることから，ひとつのパターンにつき 4 つのパターンを記述した．

記述したパターンに当てはまれば，3 章で分類したパターン番号と仮引数名が出力される．3-7，3-8 については if 文の記述はないので 1 パターンとなる．条件演算子が使われているものは 1-10，最後の NULL を示す可変長引数は 3-11 と表示させるようにした．

4.4　ツールの出力結果からの開発支援

作成したツールの出力結果からどのようなことがいえるのか示す．

- 関数ごとの関数定義

これは 3.1 節で使用したものである．定義と関数呼出が関数ごとにフォルダにまとめられている．これにより，見たい関数の定義を容易に見つけることが可能になる．

- 関数呼出に関する統計（NULLtable.txt）

この統計から関数が可変長引数であるかがわかる．また，スプレッドシートで表示させ，データを用いると NULL が使われる割合などを求めることができる．引数ごとの NULL が使われる割合も算出でき，割合が 100% であれば関数を利用する際に，その引数が NULL になることを予想できる．

- パターン検出結果（func_call_table.txt）

```
listen.c 859 alloc_listener slave 3-8
config.c 809 ap_add_module sym_name 3-7
util_expr_eval.c 813 ap_expr_exec_re pmatch 3-8
util_expr_parse.c 1536 ap_expr_make a1 3-8
util_expr_parse.c 1543 ap_expr_make a1 3-8
```

図 3　パターン検出結果の例

図 3 の出力結果から，「listen.c の 859 行目にある alloc_listener という関数の引数 NULL は仮引数名が slave でパターンは 3-8」ということがわかる．

以上の 3 つからソースコードの開発支援のための確認するべき点を表示させることができる．

5　考察

本研究では，関数呼出の NULL に注目し，NULL の使われ方をパターン化して統計情報などを出力するツールを作成した．このツールの有効性を調べるために，Apache 以外のソースコードで適用実験を行う．他のソースコードでツールを実行させ，どのくらいの割合でパターンが検出されるのか，正しくパターンを検出できるかの確認をする．検出できない関数呼出があれば，なぜ検出できなかったのか考察する．

5.1　Coreutils，nginx の適用実験

Coreutils-9.0 と nginx-1.21.4 で適用実験を行う．ツールを実行した結果，パターン検出数，割合は表 7 になった．

この検出結果が正しく検出されているか定義と見比べ確認したところ，全て正しく検出されており，分類ミスはな

表 7 CoreUtils，nginx の検出結果

	Coreutils		nginx	
パターン番号	検出数	割合	検出数	割合
1-1	38	31.93	0	0.00
1-2	0	0.00	21	9.68
1-4	9	7.56	0	0.00
1-5	0	0.00	0	0.00
1-7	0	0.00	0	0.00
1-8	26	21.85	21	9.68
2-7	21	17.65	28	12.90
2-8	0	0.00	0	0.00
3-7	2	1.68	0	0.00
3-8	20	16.81	133	61.29
1-10	3	2.52	14	6.45
3-11	0	0.00	0	0.00

かった．

続いて，このツールでどのくらいの関数呼出が検出されなかったか調べた．Coreutils で調べることができる関数呼出は全部で 516 個であり，検出した関数呼出は 119 個で未検出は 397 個であった．この未検出は調べることができる関数呼出の約 77% にあたる．nginx では調べることができる関数呼出は 254 個で，検出した関数呼出は 217 個，未検出は 37 個であった．この未検出は調べることができる関数呼出の約 15% にあたる．

5.2 未検出の原因

Coreutils では約 77%，nginx では約 15% が未検出であった．未検出の原因は以下の 4 つの原因が考えられる．

原因 1 定義中でさらに関数呼出として使われるパターン

「func(…, ptr, …)」のパターンである．3.3 節で 1-3，1-6，1-9，2-9，3-9 として分類していたが，4.3 節で TEBA での記述が難しいので自動分類対象から外した．このパターンは NULL になる実引数を定義内で NULL と書き換えることができれば，定義をさらに追うことでどのような使われ方をしているかわかる．実引数名を自動で NULL に書き換えるツールを作れば，パターン検出でき，理解支援が行えると考えられる．

原因 2 if 文の条件に論理演算が使われるパターン

これは if 文の条件が「ptr&&式」のように，パターンマッチングができないパターンである．今回 TEBA で書いたパターン記述は条件が 1 つであったため検出されない．この記述は対応する TEBA のパターン記述を追加することで検出することができるようになる．

原因 3 if 文の else 文の方で使われるパターン

これは if 文の then 部ではなく，else 部に ptr が書かれるパターンである．これについても対応する TEBA のパターン記述を追加することで検出することができるようになる．

原因 4 その他複雑な書き方をしているパターン

その他キャストなどの書き方をしており，検出できなかったものがあった．他の定義でこのような記述で書かれているものは非常に少なかったので，追加のパターン記述をしても検出できる関数の数は少ないと考えられる．

以上の適用実験から，未検出は多いものの，TEBA で記述した 10 パターンについては正しく検出されることがわかった．未検出については追加で TEBA で記述をすることで，検出できるものも増えると考えられる．

5.3 パターン分類ツールの利用

本研究ではパターン分類をするために NULL に関する情報の提示を行った．パターン分類ツールで NULL についての情報や定義をすでに抽出済みなので，これらを利用し新たなツールの作成ができる．ユーザ自身で知りたい情報を抜き出し，ツールを作ることでユーザが必要とする情報のみを提示するオリジナルの開発支援ツールが作成できる．

6 おわりに

本研究では関数呼出の NULL の調査・分類をし，それを元にパターン分類ツールの作成をした．本当に引数が NULL で良いかの確認と実引数に NULL が使える関数という情報の提示を目的とした．そのために実際に使われるオープンソースを調査し，NULL の使われ方をパターン化した．このパターン化からなぜ NULL になるのかの理由，NULL になる割合を提示した．今後の課題としては，パターン定義できていない記述の分析，ツールで未検出になるパターンが他にあるかの調査，他のリテラルでの適用が挙げられる．

参考文献

[1] Apache httpd-2.4.46, https://httpd.apache.org/

[2] Apache Portable Runtime, https://apr.apache.org/

[3] 鈴木英梨花，酒井三四郎: “静的解析結果と実行履歴を組み合わせたメソッド呼び出し可視化によるコードリーディング支援”, 第 81 回全国大会講演論文集（2019）.

[4] 太田洋介，大久保弘崇，粕谷英人，山本晋一郎: “C プログラムの理解を支援するナビゲーション機能”, 社団法人 情報処理学会 研究報告 （2004）.

[5] TEBA, http://tebasaki.jp/src/