

2024 機械情報夏学期 ソフトウェア第二

担当：岡田 慧 k-okada-soft2@jsk.t.u-tokyo.ac.jp

2. デバッグ技法

1 デバッグ技法

本章の内容は「C 実践プログラミング」¹によるところが多い。この本はCの基礎から始まり、実際にプログラミング、デバッグ、最適化、移植までカバーしており、大変良い本である。

プログラム作成において最も時間のかかる作業はデバッグであるが、デバッグには、こうしたら必ず解が見つかる、という決まった方法があるわけではない。基本的にはできるだけ多くのプログラムを書きデバッグをすることでしかスキルを向上できないだろう。しかしながら、それでも、ここではいくつかのコツを紹介する。

1.1 コメント

プログラムには必ずコメントを入れよう。プログラムを書いている時にはその構造が頭の中に入っているが、次の日にみたらまったく分からなくなることが多い。プログラムのデバッグ時には、プログラムの構造を理解するためにも、必ずコメントを書きしておくこと。

コメントを書く目安としては以下がある。

- プログラムで実現しようとしている機能の説明
- 関数の機能はなにか、引数の意味とその用途、返り値の意味
- 各制御構造の狙い、アルゴリズムの名前、仕組み
- 各変数の意味、利用方法

しかしながら、これらのことは変数名や関数名を工夫すればよいとも言える。一番重要なコメントは、なぜ、その様なコードを書いているか、という意図や選択を説明するもの。また、ある値では期待通りに動かない場合や、標準的でない使い方を想定している場合に、きちんとコメントを書きしておくべきである。

例えばファイルの内容を表示するプログラム cat のプログラムソースコードは <https://github.com/coreutils/coreutils/blob/master/src/cat.c> にある。非常に単純なプログラムであるが、丁寧なコメントが書かれていることに注目してほしい。

¹<https://t.ly/4x10W>

```

/* test0.c */
#include <stdio.h>

/*
 * 引数：整数 i, j
 * 返値：i と j の積となる整数
 * 機能：引き数の積を計算し返す関数
 */
int test(int i, int j) {
    return (i * j);
}

```

```

int main(int argc, char *argv) {
    int i, j; /* 入力となる整数 i, j */
    int k;    /* i と j の積 */

    i = 3;
    j = 2;

    /* i と j を掛けて k に代入する */
    k = test(i, j);

    /* k の値が 5 より大きければ ">5" と表示し、
     そうでなければ "<=5" を表示する */

    if (k > 5) printf(">5\n");
    else printf("<=5\n");
    /* if 文の後には曖昧性をさけるため中括弧{を入れるべきである。
     ここでは後述の演習のため{を使っていない */

    return 0;
}

```

1.2 インデント

インデント（字下げ）をきちんと記述することはとても大切なことである．インデントにより²左のプログラムは右のプログラムになり，なにかおかしいことがわかる．

```

#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
    int k = atoi(argv[1]);
    if (k > 5)
        printf(">5\n");
    else
        printf("<=5\n");
}

```

```

#include <stdio.h>

int main (int argc, char *argv[]) {
    int k = atoi(argv[1]);
    if (k > 5)
        printf(">5\n");
    else
        printf("<=5\n");
}

```

同様に，以下の例も見よう．

²emacs では各行で tab をうつとよい

```
if (k > 5)
    printf("check!!\n");
    printf(">5\n");
else
    printf("<=5\n");
```

```
if (k > 5)
    printf("check!!\n");
printf(">5\n");
else
    printf("<=5\n");
```

正しくは、

```
if (k > 5) {
    printf("check!!\n");
    printf(">5\n");
} else {
    printf("<=5\n");
}
```

であろう。

また、emacs のリージョン機能を使えば指定した領域全体のインデントを揃えることが出来る。まず指定したいリージョンの先頭の行にカーソルを動かし³、C-Space⁴、そしてリージョンの最後に移動して、M-x indent-region⁵と打つ。

1.3 プリント文デバッグ

プリント文によるデバッグ用のコードを必要に応じて挿入し、削除するためには、プリプロセッサを使う方法とコマンドラインスイッチという方法がある。

プリプロセッサとはコンパイル前にソースコードに対して処理を掛けるプログラムであり、

```
#include <stdio.h>
#include <stdlib.h>

int fact (int x) {
    if (x >= 0) {
#ifdef DEBUG
        printf("x = %d\n", x);
#endif
        return ( x * fact (x - 1) );
    } else {
#ifdef DEBUG
        printf("x = %d, return 1\n", x);
#endif
        return 1;
    }
}
```

```
int main (int argc, char *argv[]) {
    int x, ret;
    x = atoi(argv[1]);
    ret = fact(x);
    printf("ret = %d\n", ret);
}
```

とプログラムが書いてある場合、

```
$ gcc -o fact fact.c
```

としてコンパイルする場合は、`#ifdef` と `#endif` で囲まれた部分は有効にならないが、

³カーソルを C-p, C-n, C-f, C-b で動かそう。これは emacs キーバインドと呼ばれる。一方、j,k,l でカーソルを動かす物を vi 流のキーバインドと呼ぶ。emacs より難易度は高いが、gmail のキーボードショートカットは vi 流である。

⁴コントロールキーを押しながらスペースキーを押す

⁵Esc キーを押した後、x キーをおし、indent-region と打ち込む。途中で tab キーを打つと候補を教えてくれたり、補完してくれたりする。

```
$ gcc -o fact -DDEBUG fact.c
```

とすると有効になる．実際にプログラムを実行すると以下の様になる．

```
$ ./fact 10
x = 10
x = 9
x = 8
(中略)
x = 2
x = 1
x = 0
x = -1, return 1
ret = 0
```

一方，毎回再コンパイルするのは面倒だとなった場合以下の様にコマンドラインスイッチを使うと良い．`argc`, `argv` を使い⁶，

```
#include <stdio.h>
#include <stdlib.h>

#define FALSE 0
#define TRUE 1

int Debug = FALSE;

int fact (int x) {
    if (x > 0) {
        if ( Debug ) {
            printf("x = %d\n", x);
        }
        return ( x * fact (x - 1) );
    } else {
        if ( Debug ) {
            printf("x = %d, return 1\n", x);
        }
        return 1;
    }
}
```

```
int main (int argc, char *argv[]) {
    int x, ret;
    while ( ( argc > 1 ) && (argv[1][0] == '-') ) {
        switch (argv[1][1]){
            case 'd': Debug=TRUE; break;
        }
        argc--;argv++;
    }
    x = atoi(argv[1]);
    ret = fact(x);
    printf("ret = %d\n", ret);
}
```

のようにし，

```
$ gcc -o fact fact.c
```

としてコンパイルすると，`-debug` をつけるかつかないかで以下の様にプログラムの振る舞いを変えることが出来る．

```
$ ./fact -debug 10
x = 10
x = 9
x = 8
(中略)
x = 2
x = 1
x = 0, return 1
3628800
$ ./fact 10
3628800
```

⁶夏学期の計算機演習では，より便利なライブラリ `getopt` の使い方を教える予定である．

1.4 対話型デバッガを利用したデバッグ

C 言語ではデバッグ用のツールが用意されている。これをデバッガと呼ぶ。ここでは、

```
$ ./fact
Segmentation fault
```

となる事例をデバッグしてみよう。

まず、プログラムにデバッグ情報を埋め込むため以下の様に `-g` オプションをつけてしてコンパイルする⁷。

```
$ gcc-7 -o fact -g fact.c
```

デバッガは実行ファイル名を引数に以下の様に呼び出す。

```
$ gdb ./fact
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
(中略)
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./fact...
(gdb)
```

最後に表示されている (gdb) というのがプロンプトである。run というコマンドで実行できる。

```
(gdb) run
Starting program: fact

Program received signal SIGSEGV, Segmentation fault.
__GI___strtol_l_internal (nptr=0x0, endptr=endptr@entry=0x0, base=base@entry=10,
group=group@entry=0, loc=0x7ffff7f7b4a0 <_nl_global_locale>) at ../stdlib/strtol_l.c:292
292 ../stdlib/strtol_l.c: No such file or directory.
(gdb)
```

この状態ではいくつかのコマンドが利用できる。

```
(gdb) backtrace
#0 __GI___strtol_l_internal (nptr=0x0, endptr=endptr@entry=0x0, base=base@entry=10,
group=group@entry=0, loc=0x7ffff7f7b4a0 <_nl_global_locale>) at ../stdlib/strtol_l.c:292
#1 0x00007ffff7dd6ab6 in __strtol (nptr=<optimized out>, endptr=endptr@entry=0x0,
base=base@entry=10) at ../stdlib/strtol.c:106
#2 0x00007ffff7dd25c4 in __GI_atoi (nptr=<optimized out>) at atoi.c:27
#3 0x0000555555555238 in main (argc=1, argv=0x7fffffffe508) at fact.c:30
```

backtrace または bt コマンドにより、プログラムが main からどのように呼び出され、現在の状態に至ったかを知る、バックトレースを行うことができる。main 関数から atoi 関数がよばれ、その先の strtol 関数でプログラムが停止していることがわかる。

⁷原因は調査中だがデフォルトの gcc-9 でコンパイルすると break する場所が一行異なっている。原因は調査中。clang だと期待通り（？あるいは以前の？）の挙動になる。

```
(gdb) up
#1 0x00007ffff7dd6ab6 in __strtol (nptr=<optimized out>, endptr=endptr@entry=0x0,
    base=base@entry=10) at ../stdlib/strtol.c:106
106 ../stdlib/strtol.c: No such file or directory.
(gdb) up
#2 0x00007ffff7dd25c4 in __GI_atoi (nptr=<optimized out>) at atoi.c:27
27 atoi.c: No such file or directory.
(gdb) up
#3 0x0000555555555238 in main (argc=1, argv=0x7fffffffe508) at fact.c:30
30  x = atoi(argv[1]);
```

up コマンドによりフレームをたどっていき、fact.c の 30 行め、atoi のところが問題だとわかる。

```
(gdb) print argv[1]
$1 = 0x0
```

とすると、変数の値を見ることが出来る。argv[1] は引数の文字列を想定しているのに 0x0(null)が入っている。

よく考えると、引数で階乗したい数値を入れる必要があった。引数をつけてプログラムを実行するには run 引数とすればよい。

```
(gdb) run 10
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /fact 10
ret = 3628800
```

となった。

また、プログラムを実行中に停止するブレイクポイントも break 関数名 または、b 関数名 コマンドを用いて設定できる。

```
(gdb) break fact
Breakpoint 1 at 0x555555555175: file fact.c, line 10.
```

として設定し、

```
(gdb) run 10
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /fact 10

Breakpoint 1, fact (x=10) at fact.c:10
10  if (x > 0) {
```

とすると設定したブレイクポイントで停止する。

print 変数名 コマンドを用いて

```
(gdb) print x
$1 = 10
```

とすると変数を見ることができ、next または n コマンドを用いて

```
(gdb) next
11     if ( Debug ) {
(gdb) n
14     return ( x * fact (x - 1) );
```

とするとステップ実行が可能になる。

また、似たようなコマンドに step コマンドがある。step は関数呼び出し時に呼び出し先の関数の内部に渡ってトレースをかける点異なる。

list コマンドを用いることで現在のステップ周辺のソースコードを見ることができる。

```
(gdb) list
5 #define TRUE 1
6
7 int Debug = FALSE;
8
9 int fact (int x) {
10     if (x > 0) {
11         if ( Debug ) {
12             printf("x = %d\n", x);
13         }
14         return ( x * fact (x - 1) );
```

continue または c コマンドを用いて

```
(gdb) continue

Breakpoint 1, fact (x=9) at fact.c:10
10     if (x > 0) {
(gdb) c
Continuing.

Breakpoint 1, fact (x=8) at fact.c:10
10     if (x > 0) {
```

プログラムの処理を続けることができる。ここでは、fact 関数に break を掛けているため、関数が呼ばれる度にプログラムが停止している。

他にも、info breakpoint でブレイクポイントを確認し、delete でブレイクポイントを取り消すことができる。さらに、break ファイル名:行数で任意の位置にブレイクポイントを指定したり、break ファイル名:行数 if 条件で、与えられた条件を満たした際にのみブレイクを指定できる。

```
(gdb) info breakpoint
Num      Type      Disp Enb Address          What
1        breakpoint keep y  0x000055555555175 in fact at fact.c:10
breakpoint already hit 1 time
(gdb) delete 1
(gdb) break fact.c:14 if x <= 2
Breakpoint 2 at 0x5555555519b: file fact.c, line 14.
(gdb) info breakpoint
Num      Type      Disp Enb Address          What
2        breakpoint keep y  0x00005555555519b in fact at fact.c:14
stop only if x <= 2
```

この状態で実行すると以下のように引数 2 で fact が呼びされて初めて停止する .

```
(gdb) run 10
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: fact 10

Breakpoint 2, fact (x=2) at fact.c:14
14      return ( x * fact (x - 1) );

(gdb) c
Continuing.

Breakpoint 2, fact (x=1) at fact.c:14
14      return ( x * fact (x - 1) );
(gdb) c
Continuing.
ret = 3628800
[Inferior 1 (process 151078) exited normally]
\begin{verbatim}
```

また , set コマンドで実行中に変数の値を変えたり , watch コマンドで変数が変更された情報を表示したり , disassemble 関数名または disass 関数名コマンドでアセンブラを表示するなど , 多様な機能を有している .

```
(gdb) run 10
Starting program: fact 10

Breakpoint 2, fact (x=10) at fact.c:10
10  if (x > 0) {
1:  x = 10
(gdb) set x = 0
(gdb) set Debug = 1
(gdb) n
16      if ( Debug ) {
(gdb) n
17          printf("x = %d, return 1\n", x);
(gdb) n
x = 0, return 1
19      return 1;
(gdb) n
21 }
(gdb) n
main (argc=2, argv=0x7fffffff4f8) at fact.c:32
32  printf("ret = %d\n", ret);
(gdb) c
Continuing.
ret = 1
[Inferior 1 (process 151091) exited normally]
```

以下のプログラムは繰り返しによる階乗のプログラムであり , これまで紹介してきた fact.c とは異なっていることに注意 .


```
(gdb) b fact
Breakpoint 1 at 0x1171: file fact-loop.c, line 10.
(gdb) run 10
Starting program: fact-loop 10

Breakpoint 1, fact (x=10) at fact-loop.c:10
10  int r = 1;
(gdb) list
5  #define TRUE 1
6
7  int Debug = FALSE;
8
9  int fact (int x) {
10  int r = 1;
11  while (x > 0) {
12      r = r * x;
13      x = x - 1;
14  }
(gdb) watch r
Hardware watchpoint 2: r
(gdb) c
Continuing.

Hardware watchpoint 2: r

Old value = 32767
New value = 1
fact (x=10) at fact-loop.c:11
11  while (x > 0) {
(gdb) c
Continuing.

Hardware watchpoint 2: r

Old value = 1
New value = 10
fact (x=10) at fact-loop.c:13
13      x = x - 1;
(gdb) c

(中略)

Hardware watchpoint 2: r

Old value = 1814400
New value = 3628800
fact (x=2) at fact-loop.c:13
13      x = x - 1;
(gdb) c
Continuing.

Watchpoint 2 deleted because the program has left the block in
which its expression is valid.
0x000055555555203 in main (argc=2, argv=0x7fffffffe4f8) at fact-loop.c:26
26  ret = fact(x);
(gdb) c
Continuing.
ret = 3628800
[Inferior 1 (process 151273) exited normally]
```

```
(gdb) disassemble fact
Dump of assembler code for function fact:
0x000055555555516a <+0>: push    %rbp
0x000055555555516b <+1>: mov     %rsp,%rbp
0x000055555555516e <+4>: sub     $0x10,%rsp
0x0000555555555172 <+8>: mov     %edi,-0x4(%rbp)
0x0000555555555175 <+11>: cmpl    $0x0,-0x4(%rbp)
```

その他の使い方はインターネット上のマニュアルを参照されたい．公式の情報は <https://www.gnu.org/software/gdb/documentation/> にあり，また日本語訳は例えば http://flex.phys.tohoku.ac.jp/texti/gdb-j/gdb-j_toc.html から得ることが出来る．

1.5 関数トレースツール

覚えておくと良いデバッグツールとして関数トレースツールがある．代表的なものにユーザが作成した関数を含むライブラリ関数をトレースする `ltrace` と，OS(正確には OS のカーネル) の機能呼び出すのに利用されるシステムコールをトレースする `strace` がある．

1.5.1 ライブラリ関数トレースツール (`ltrace`) の使い方

実行されたライブラリ関数を表示するトレースツールとして `ltrace` がある．簡単には以下のようになり，プログラムの前に `ltrace` をつけて実行すれば良い⁸．

```
$ ltrace ./fact 5
atoi(0x7ffc401d910d, 0x7ffc401d81d8, 0x7ffc401d81f0, 0x55fa7026d730) = 5
printf("x = %d\n", 5x = 5
)                                     = 6
printf("x = %d\n", 4x = 4
)                                     = 6
printf("x = %d\n", 3x = 3
)                                     = 6
printf("x = %d\n", 2x = 2
)                                     = 6
printf("x = %d\n", 1x = 1
)                                     = 6
printf("x = %d, return 1\n", 0x = 0, return 1
)                                     = 16
printf("ret = %d\n", 120ret = 120
)                                     = 10
+++ exited (status 0) +++
```

また `man ltrace` とすると様々なオプションを見ることが出来る．

例えば，プログラムを2つのソースコードに分けて，関数部分を `gcc -shared` でコンパイルして動的ライブラリ化しリンクする．

⁸原因は調査中だがデフォルトの `gcc-9` でコンパイルすると表示されない．原因は調査中．`gcc-7` か，`clang` だと期待通り(？あるいは以前の？)の挙動になる．

```
/* fact-main.c */

#include <stdio.h>
#include <stdlib.h>

extern int fact (int x);

int main (int argc, char *argv[]) {
    int x, ret;
    x = atoi(argv[1]);
    ret = fact(x);
    printf("ret = %d\n", ret);
}
```

```
/* fact-lib.c */

#include <stdio.h>
#include <stdlib.h>

int fact (int x) {
    if (x > 0) {
#ifdef DEBUG
        printf("x = %d\n", x);
#endif
        return ( x * fact (x - 1) );
    } else {
#ifdef DEBUG
        printf("x = %d, return 1\n", x);
#endif
        return 1;
    }
}
```

これらのソースコードを準備し以下のようにして動的ライブラリを生成する .

```
all: fact-main

libfact-lib.so:
    gcc-7 -shared -fPIC -DDEBUG -o libfact-lib.so fact-lib.c

fact-main: libfact-lib.so
    gcc-7 -o fact-main fact-main.c -Wl,-rpath,$(PWD) -L$(PWD) -lfact-lib
```

最後に以下のように実行することで fact-lib.c 内の fact 関数が呼び出されたことを確認出来る .

```
$ ltrace -T -l libfact-lib.so ./fact-main 5
fact-main->fact(5, 0x7ffe35932108, 0, 0x1999999999999999x = 5
<unfinished ...>
libfact-lib.so->fact(4, 0x556077063260, 0, 0x = 4
<unfinished ...>
libfact-lib.so->fact(3, 0x556077063260, 0, 0x = 3
<unfinished ...>
libfact-lib.so->fact(2, 0x556077063260, 0, 0x = 2
<unfinished ...>
libfact-lib.so->fact(1, 0x556077063260, 0, 0x = 1
<unfinished ...>
libfact-lib.so->fact(0, 0x556077063260, 0, 0x = 0, return 1
)    = 1 <0.000361>
<... fact resumed> )    = 1 <0.000929>
<... fact resumed> )    = 2 <0.001469>
<... fact resumed> )    = 6 <0.002047>
<... fact resumed> )    = 24 <0.002640>
<... fact resumed> )    = 120 <0.003650>
ret = 120
+++ exited (status 0) +++
```

1.5.2 システムコールトレースツール (strace) の使い方

また、OS(正確には OS のカーネル) の機能呼び出すのに利用されるシステムコールをトレースする `strace` がある。デバイス関係のプログラムなどよりシステムを直接操作するようなプログラムをデバッグする際には有用であるので覚えておくと良い。

```
$ strace ./fact 5
execve("./fact", [ "./fact", "5"], 0x7fffae668418 /* 69 vars */) = 0
brk(NULL)                               = 0x555600e45000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=384359, ...}) = 0
mmap(NULL, 384359, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5652435000
close(3)                                 = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f5652433000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5651e7b000
mprotect(0x7f5652062000, 2097152, PROT_NONE) = 0
mmap(0x7f5652262000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f5652262000
mmap(0x7f5652268000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5652268000
close(3)                                 = 0
arch_prctl(ARCH_SET_FS, 0x7f56524344c0) = 0
mprotect(0x7f5652262000, 16384, PROT_READ) = 0
mprotect(0x5555ff57d000, 4096, PROT_READ) = 0
mprotect(0x7f5652493000, 4096, PROT_READ) = 0
munmap(0x7f5652435000, 384359)          = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 7), ...}) = 0
brk(NULL)                               = 0x555600e45000
brk(0x555600e66000)                     = 0x555600e66000
write(1, "x = 5\n", 6x = 5
)                                         = 6
write(1, "x = 4\n", 6x = 4
)                                         = 6
write(1, "x = 3\n", 6x = 3
)                                         = 6
write(1, "x = 2\n", 6x = 2
)                                         = 6
write(1, "x = 1\n", 6x = 1
)                                         = 6
write(1, "x = 0, return 1\n", 16x = 0, return 1
)                                         = 16
write(1, "ret = 120\n", 10ret = 120
)                                         = 10
exit_group(0)                           = ?
+++ exited with 0 +++
```

1.6 告白的方法によるデバッグ

「C 実践プログラミング」には「15.5 告白的方法によるデバッグ」としておもしろいことが書いてある。少々長いが引用しよう。

告白的方法によるデバッグとは、プログラマが、作成したプログラムについて他人に説明することで行うデバッグです。説明する相手は、関係者でも構いませんし、関係者でなくても構いません。ただの壁に向かって説明しても構いません。プログラマが話し掛けることが出来さえすれば誰でも構わないのです。

典型的な対話例は次のようなものです。

「やあ、ビルちょといいかな。どうやら僕が作ったプログラムにはバグがあるようなんだ。8.0 と出力されるはずなんだけど、実際は-8.0 となってしまうんだ。出力する値はこの式を使って計算しているんだけどー支払い金額とレートはチェックしたし、閏年コードがおかしくなければ日付は正しいはずだし、そのコード、ん… ありがとう、ビル、問題点がわかったよ。」

その間、ビルは一言も発しませんでした。

このようなデバッグは立ち稽古とも呼ばれています。他者の存在により、違った方向からプログラムを眺めることができるようになります。さらに、本人が見過ごしていた問題点を愛絵が指摘してくれることもまれではありません。

これは、大変有効な方法で他人にどういうバグがあるか、ということを伝えようとする、いろいろな方向から説明する必要がある。そういうして視点を変えている内に自分で問題が分かってしまうことがよくある。ぜひ、まわりの人⁹にバグを報告し、解決して欲しい。

1.7 バグの報告

バグの報告方法には決まった形式はなく¹⁰、そもそも適切にバグを報告できるようになれば、ほぼ問題を解決できるので、まずは分かる範囲で報告 / 説明するのが良い。慣れてきたら以下の点に気をつけよう¹¹。

- 再現できるようにそこに至った経緯も含め、できる限り具体的に書くこと。どういう出力を期待していて、現状どうなっているか。
- あなたが何を目的に何をしたのかを書き、結果として何が問題なのかも厳密に述べること。“XXX がうまくいきません。” だけでは手伝うことが困難である。また質問の題に問題の簡潔な描写も含めること。
- warnings と errors の情報を必ず含めること。ターミナルからそれらを直接コピーアンドペーストすること。決して自分で warnings や errors をタイプしないよう。小さなタイプミスが大きな違いと時間の無駄を生み出すことがある。

⁹もちろん k-okada-soft2@jsk.t.u-tokyo.ac.jp に報告してもらっても構わない。

¹⁰「技術系メーリングリストで質問するときのパターン・ランゲージ」という有名な文章があるが初心者にはハードルが高いだろう

¹¹<http://wiki.ros.org/ja/Support>

2023/04/09 17:52

動かないhello-world.c

三

New chat

+

 hello-world.c が動きません。



それは残念ですね。問題が何であるかを判断するには、詳細が必要です。

例えば、何かエラーメッセージが表示されますか？もしそうなら、そのメッセージを教えてください。また、どのような環境でプログラムを実行していますか？どのようなコマンドを使用してプログラムをコンパイルおよび実行しようとしていますか？

可能であれば、問題のコードも共有していただくと助かります。それによって、私たちはあなたの問題をより具体的に理解し、より正確な回答を提供することができます。

Send a message...

[ChatGPT Mar 23 Version](#). Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts

<https://chat.openai.com/chat>

1/1

2 宿題

提出先：ITC-LMS を用いて提出すること

提出内容：以下の問題の実行結果の画面をキャプチャしファイル名は「問題番号.png」とし、また講義中にでてきたキーワードについて知らなかったもの、興味のあるものを調べ「学籍番号.txt」としてアップロードすること。テキストファイルはワードファイルなどだと確認出来ないことがあるため、emacs/vi等のテキストエディタを使って書こう。プログラムが長くなりキャプチャ画面に入り切らなくなってきたらプログラムファイルと実行結果を「問題番号.txt」にまとめてアップロードしてよい。

画像で提出する場合は、各自のマシンのMacアドレスが分かるようにすること。例えば画面中にifconfigというコマンドを打ち込んだターミナルを表示すればよい。

ITC-LMSにアップロードする際には講義・宿題の感想を必ずコメントに記すこと。また授業中に質問した者はその旨を記すこと。質問は成績評価時の加点対象となる。

キーワード：ブレークポイント、ステップ実行、スタックトレース、システムコール、共有ライブラリ

1. test0.c, fact.c 等事業で扱ったプログラムや、その他（ソフトウェア演習等）で各自が書いたプログラムにコメントをつけよ。
2. 対話型デバッガを利用してプログラムを実行してみよ。適宜ブレークポイントを入れて、変数を確認してみよう。
3. （発展課題）ソフトウェア演習等でバグに遭遇した際には対話型デバッガ、あるいはトレースツールを用いてみよ。その過程でバグが見つければ、どのように判断したか記せ。

A ChatGPT による回答例


2023/04/09 18:03

New chat

+

☰

New chat



以下のプログラムにコメント文を追加してください

```
#include <stdio.h>
#include <stdlib.h>

int fact (int x) {
    if (x >= 0) return (x * fact (x - 1));
    else      return 1;
}

int main (int argc, char *argv[]) {
    int x, ret;
    x = atoi(argv[1]);
    ret = fact(x);
    printf("ret = %d\n", ret);
}
```

< 5 / 5 >

✎

```
// C言語の標準入出力ライブラリをインクルードする
#include <stdio.h>
// 標準ライブラリ関数を使用するためにインクルードする
#include <stdlib.h>

// 関数factを定義する
// x: 整数値
int fact (int x) {
    // xが0以上の場合、再帰的に自身を呼び出して階乗を計算する
    if (x >= 0) {
        return (x * fact (x - 1));
    }
    // xが負の場合、0!として1を返す
    else {
        return 1;
    }
}
```

⬇

Send a message...

📎

🔄

ChatGPT Mar 23 Version. Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts

<https://chat.openai.com/chat>

1/1