

# ソフトウェアテスト

## [7] ホワイトボックステスト演習

---

Software Testing  
[7] White Box Testing Exercise & Seminar

あまん ひろひさ  
阿萬 裕久 (AMAN Hirohisa)  
aman@ehime-u.ac.jp

(C) 2007-2024 Hirohisa AMAN

1

## 演習の目的

---

- プログラムの**ホワイトボックステスト**を行い、  
ブラックボックステストとあわせてソフトウェア  
テストの重要性を体験・学習する
- 主な内容
  - ツールを使った命令網羅の確認
  - 仕様に従ったプログラムの作成
  - 他人によるテストの実施
  - (不具合が見つかった場合は)プログラムの修正

(C) 2007-2024 Hirohisa AMAN

2

# 演習の内容

## □ 課題1

与えられたプログラム(バグあり)に対して**テストを行い**, 命令網羅の確認を行う(ツールを利用)

## □ 課題2

与えられた**仕様に従ってプログラムを作成**する  
そして,  
**作成したプログラムを他人にテストしてもらう**  
**不具合が見つかった場合はプログラムの修正**も行う

## 課題1

第 5 回の演習  
([05] Exercise)  
で使ったものと同じ

【テスト対象】 **ex0501.c**

- ① **[5] Exercise-1 で作った**テストケース(**test-report0501.xlsx**)を**test-report0701.xlsx にコピー**しなさい。そして、それらを順番に実行して命令網羅率を調べなさい。
- ② テストケースをすべて実行しても**命令網羅率が100%にならない場合**は、**テストケースを追加**しなさい(**test-report0701.xlsx に追加**しなさい)

# 課題1で使うツール

---

- **gcov** というツール(プログラム)を使用する
  - これは通常, gcc と一緒にインストールされている
  - このツールは gcc でコンパイルされたソースファイルと実行ファイルを調べ, **各命令が何回実行されたのかを記録**できる

## gcov 利用の流れ(1/7)

---

- ①まずはプログラムを gcc でコンパイルする  
ただし, **-coverage** というオプションを付ける

  
`gcc -coverage ex0501.c`

- ②そして, プログラムを実行して, まずは 1 つ目のテストケースについてテストを行う

例えば

- 10分駐車 (parking time = 10)
  - 買い物額 0 円 (purchase = 0)
- の場合を実行

## gcov 利用の流れ(2/7)

Mac 環境の人は gcov で出力される命令網羅率が説明で使っている例と少し異なる場合があるようです(命令としてカウントするものの定義が少し異なるため)。

③次に gcov コマンドで実行結果の解析とソースプログラムとの対応付けを行う

```
gcov ex0501.c
```

```
File 'ex0501.c'  
Lines executed:73.53% of 34  
Creating 'ex0501.c.gcov'
```

この行に  
**命令網羅率**が  
表示される  
(73.53%)

詳細を記録した  
ex0501.c.gcov  
というファイルが作られる

(C) 2007-2024 Hirohisa AMAN

7

## gcov 利用の流れ(3/7)

Mac 環境の場合は以下の例とは表示が少し異なっている可能性があります

④ ex0501.c.gcov の内容を見る

```
1  -: 0:Source:sample0501.c  
2  -: 0:Graph:sample0501.gcov  
3  -: 0:Data:sample0501.gcov  
4  -: 0:Runs:1  
5  -: 1:#include <stdio.h>  
6  -: 2:  
7  1: 3:int main(void){  
8  -: 4: int time, amount, sale, discount, bill, t;  
9  -: 5:  
10 1: 6: printf("Input the parking time (min): ");  
11 1: 7: scanf("%d", &time);  
12 1: 8: if ( time < 0 ){  
13 #####: 9: printf("[ERROR] invalid input\n");  
14 #####: 10: return 1;  
15 -: 11: }
```

実行回数

この部分  
に注目

ここから下  
がプログラムの  
内容

(C) 2007-2024 Hirohisa AMAN

8

## gcov 利用の流れ(4/7)

Mac 環境の場合は以下の例とは表示が少し異なっている可能性があります

### 命令網羅の情報

1	-:	0:Source:sample0501.c
2	-:	0:Graph:sample0501.gcno
3	-:	0:Data:sample0501.gcda
4	-:	0:Runs:1
5	-:	1:#include <stdio.h>
6	-:	2:
7	1:	3:in
8	-:	4:
9	-:	5:
10	1:	6:
11	1:	7:
12	1:	8:
13	#####:	9:
14	#####:	10: return 1;
15	-:	11: }

記号	意味
-	命令としては数えない
数字	その行の総実行回数
#####	その行を実行していない

(C) 2007-2024 Hirohisa AMAN

9

## gcov 利用の流れ(5/7)

Mac 環境の場合は以下の例とは表示が少し異なっている可能性があります

### ⑤次に 2 番目のテストケースで実行を行う

例えば

-10分駐車 (parking time = -10) でエラーにする

### ⑥再び gcov でチェック

**gcov** ex0501.c

命令網羅率が増えている  
(73.53% → 79.41%)

File 'ex0501.c'  
Lines executed:79.41% of 34  
Creating 'ex0501.c.gcov'

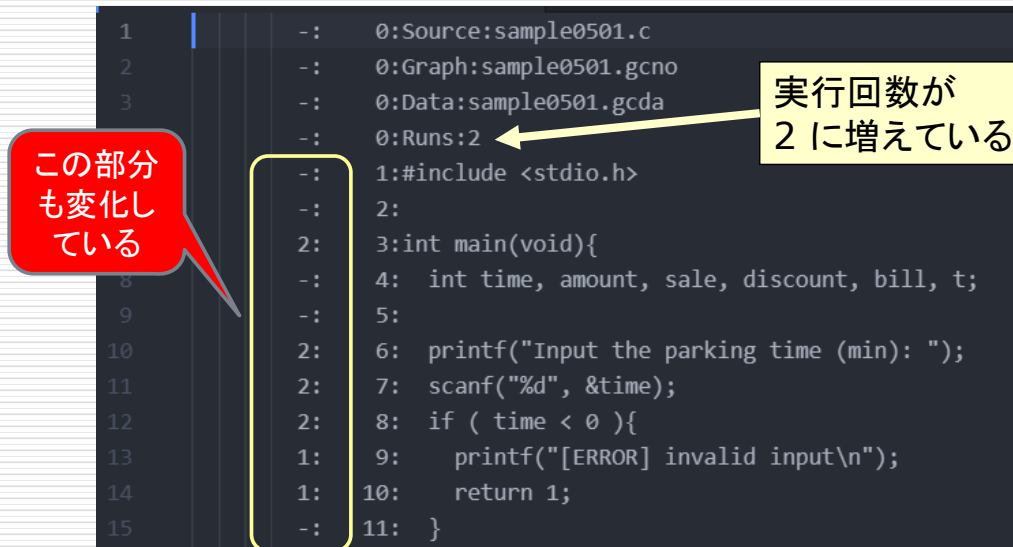
(C) 2007-2024 Hirohisa AMAN

10

## gcov 利用の流れ(6/7)

Mac 環境の場合は以下の例とは表示が少し異なっている可能性があります

### ⑦もう一度 ex0501.c.gcov の内容を見る



The screenshot shows the content of the gcov file. The first three lines are metadata: Source, Graph, and Data. The 'Runs' field is highlighted with a yellow box and an arrow pointing to it, with a callout stating '実行回数が2に増えている' (Execution count has increased to 2). The source code is listed below, with line numbers on the left. A red callout points to the first line of the source code, stating 'この部分も変化している' (This part is also changing).

```
1  -: 0:Source:sample0501.c
2  -: 0:Graph:sample0501.gcno
3  -: 0:Data:sample0501.gcda
4  -: 0:Runs:2
5  -: 1:#include <stdio.h>
6  -: 2:
7  2: 3:int main(void){
8  -: 4: int time, amount, sale, discount, bill, t;
9  -: 5:
10 2: 6: printf("Input the parking time (min): ");
11 2: 7: scanf("%d", &time);
12 2: 8: if ( time < 0 ){
13 1: 9:     printf("[ERROR] invalid input\n");
14 1: 10:    return 1;
15 -: 11: }
```

(C) 2007-2024 Hirohisa AMAN

11

## gcov 利用の流れ(7/7)

- ⑤～⑦を繰り返すことで徐々に命令網羅率が上がっていく(累積になっていることに注意)
  - ただし、テストケースによって命令網羅率の上昇の様子は異なる

### 【注意！】

gcc でのコンパイルをやり直すと、実行回数のカウントはリセットされてしまうのでくれぐれも注意すること

(C) 2007-2024 Hirohisa AMAN

12

## 課題1で提出するテスト結果

	A	B	C	D	E	F	G	H	I	J
1		input		expected result						
2	No.	parking time	purchase amout	(a) fee based on time	(b) discount	(*) total	test result	tester	failure description	statement coverage (%)
3	Ex1	1	0	100	0	100	Pass	Hirohisa Aman		73.53
4	Ex2	0	0	ERROR	ERROR	ERROR	FAIL	Hirohisa Aman	No ERROR message was displayed	79.41

第5回の課題1のシートに「statement coverage」の列が追加してある。

まずは第5回の演習で作った内容(test-report0501.xlsx)をコピーして、命令網羅率を上から順に埋めていきなさい。

もしも100%にならない場合はテストケースを追加すること(追加した行が分かるよう色を変えること)

## 課題1の提出

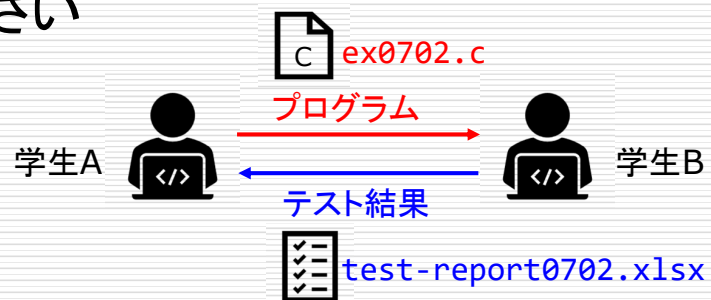
- Teams から  
test-report0701.xlsx  
を提出しなさい

### [07] Exercise-1 (white box testing)

- 提出×切: 次の講義開始時  
**13:00, Apr.22** (ベトナム時間)

## 課題2

- `ex0702.c` を完成させなさい  
(プログラムの仕様 `spec0702.pdf`)
- そして、**自分が作ったプログラムを他の人にテスト**してもらいなさい



## 課題2の手順(1/2)

ex7-materials.zip に  
含まれているが未完成に  
なっている

1. プログラム `ex0702.c` を自分で作る.  
(仕様は `spec0702.pdf`)
2. そのプログラムの**テストを他の人に依頼**する.
  - 2-1) `test-report0702.xlsx` に自分の名前とテストを行う人の名前を入力する.
  - 2-2) `ex0702.c` と `test-report0702.xlsx` をテストを行う人に渡す.



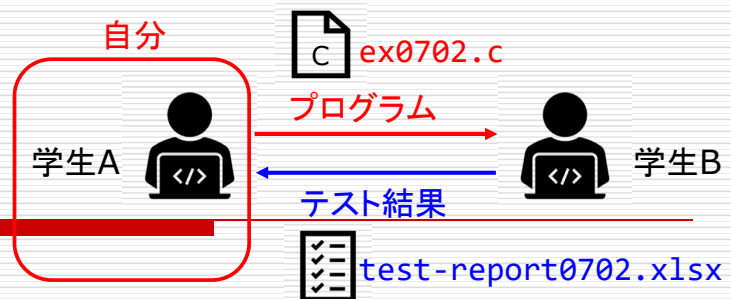
## 課題2の手順(2/2)

3. プログラムをテストして, その結果を `test-report0702.xlsx` に入力して, プログラム作成者へ返す.
4. プログラム作成者はテスト結果を確認し, バグがあればプログラムを修正して, テストをやり直す.
5. プログラム作成者は, `ex0702.c` と `test-report0702.xlsx` を Teams から提出する.

(C) 2007-2024 Hirohisa AMAN

17

## 課題2の提出



### ☐ Teams から

- 自分の作った `ex0702.c` (最終版)
- 自分が受け取った `test-report0702.xlsx` を提出しなさい

## [07] Exercise-2 (white box testing)

- ☐ 提出 ✕ 切: 次の講義開始時  
**13:00, Apr.22** (ベトナム時間)

(C) 2007-2024 Hirohisa AMAN

18

**重要**

次回の講義では R と RStudio を  
使ったデータ分析を行います

---

全員 PC を持ってくること。  
そして、R と RStudio が必要なので、  
必ずインストールしておくこと。