ソフトウェアテスト
[6] ホワイトボックステスト

Software Testing
[6] White Box Testing Techniques

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

テストの分類

- **ブラックボックステスト** 第4回で説明，第5回に演習
  - プログラムの中身は見ないもの（ブラックボックス）として，**仕様に基づいた**動作テストを行う
- **ホワイトボックステスト**
  - プログラムの**内部構造（主にフローチャート）に基づいた**動作テストを行う
- **ランダムテスト**
  - **テストケースをランダムに（無作為に）作成**して動作テストを行う

ブラックボックステスト：black box testing
仕様に基づいた（しようにもとづいた）：based on the specification
ホワイトボックステスト：white box testing
内部構造（ないぶこうぞう）：internal structure
フローチャート：flowchart
ランダムテスト：random testing
無作為に（むさくいに）：at random

ホワイトボックステスト法

- 仕様ではなく**ソースプログラムの構造（中身）**に注目してテストケースを設計する方法
  - **要求仕様に沿ったテストではない**
  - ソースプログラムでの構造の複雑さ（条件の組合せ等）に対処するもの
  - **ブラックボックステストを補足するテスト**

構造（こうぞう）：structure
中身（なかみ）：contents
構造の複雑さ（ふくざつさ）：structural complexity
条件（条件）の組合せ（くみあわせ）：combination of conditions
対処する（たいしょする）：deal with, address
補足する（ほそくする）テスト：supplemental tests, additional tests

ホワイトボックステスト法(1)
命令網羅法

- **すべての命令文を1回以上実行**する
  - 1個のテストケース(実行パス)だけでは無理な場合もある
  - しかし，**異なるテストケース**をいくつか実行し，その**集合**を考えれば網羅は可能
  - **実行できた命令の割合**を**命令網羅率**という
  - これを **C0** ともいう

命令網羅法(めいれいもうらほう):statement coverage method
命令文(めいれいぶん):statement
集合(しゅうごう):set, collection
割合(わりあい):rate

テスト対象の例(1)

```
void foo(int x){
  int sum, n;
  sum = 0;
  for ( n = 1; n < x; n++ ){
    sum += n;
  }
  printf("%d\n", sum);
}
```

【入力】
引数 x

【出力】
printf で出力される値

x > 1 ならば，すべての命令を実行できる
（命令網羅率＝100%）

テストケースの例
入力 x = 2，出力 sum = 3

引数（ひきすう）：arguments
すべての命令（めいれい）: all statements
実行（じっこう）: execution
命令網羅率（めいれいもうらりつ）: statement coverage

テスト対象の例(2)

```
void foo(int x, int y){
  int sum, n;
  sum = 0;
  for ( n = 0; n < x; n++ ){
    sum += n;
  }
  if ( sum < y ){
    printf("%d\n", sum);
  }
  else{
    printf("%d\n", y);
  }
}
```

【入力】
引数 x, y

【出力】
printf で出力される値

引数(ひきすう):arguments

命令網羅の例: x = 1, y = 0

ソースコード	実行
void foo(int x, int y){	---
int sum, n;	○
sum = 0;	○
for (n = 0; n < x; n++){	○
sum += n;	○
}	---
if (sum < y){	○
printf("%d\n", sum);	×
}	---
else{	---
printf("%d\n", y);	○
}	---
}	---

命令網羅率(C0)
$\frac{6}{7} \fallingdotseq 85.7\%$

← sum = 0 なので

【注意】
あらかじめ「どれを実行可能な命令として数えるか」を定義しておく必要がある

(C) 2007-2023 Hirohisa AMAN 7

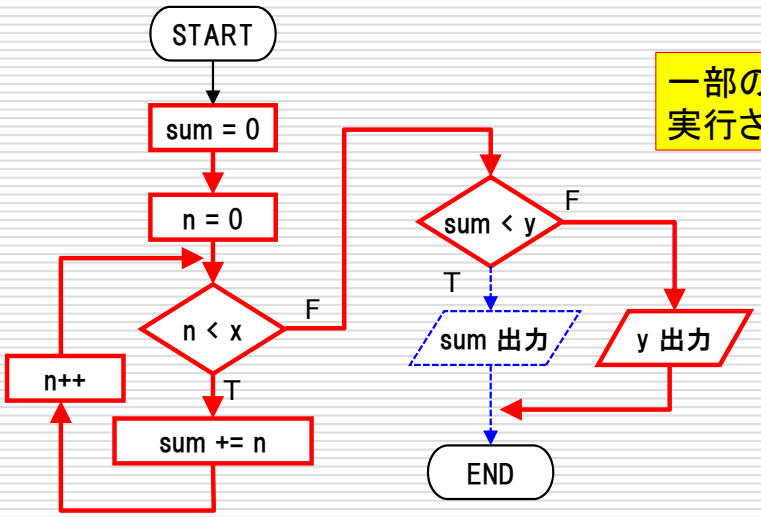
実行可能な命令(じっこうかのうなめいれい):executable statement
定義(ていぎ): define

命令網羅率 100% の例

ソースコード	x=1, y=0	x=1, y=1	total
`void foo(int x, int y){`	---	---	---
`  int sum, n;`	○	○	○
`  sum = 0;`	○	○	○
`  for ( n = 0; n < x; n++ ){`	○	○	○
`    sum += n;`	○	○	○
`  }`	---	---	---
`  if ( sum < y ){`	○	○	○
`    printf("%d\n", sum);`	×	○	○
`  }`	---	---	---
`  else{`	---	---	---
`    printf("%d\n", y);`	○	×	○
`  }`	---	---	---
`}`	---	---	---

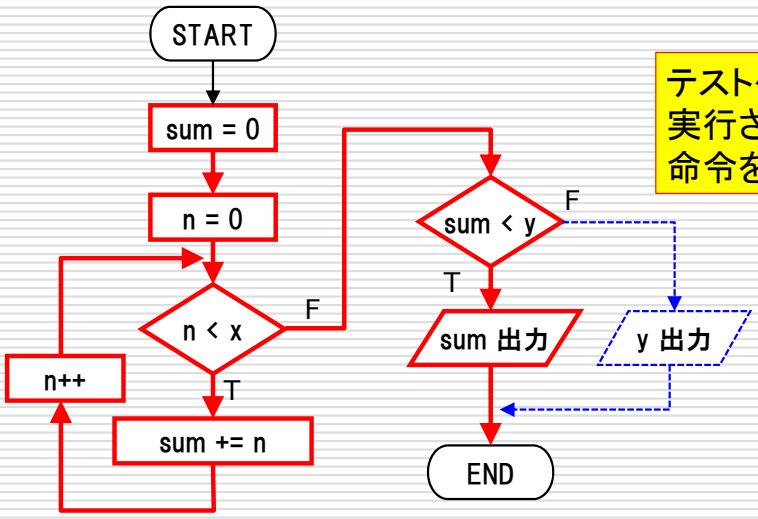
命令網羅率（めいれいもうらりつ）：statement coverage

フローチャートを描いた場合
テストケース① x = 1, y = 0



一部の命令が
実行されていない

フローチャートを描いた場合
テストケース② x = 1, y = 1



テストケース①で実行されなかった命令をカバー

命令網羅率の自動測定

- 自分で命令の実行をチェックして数えるのは**簡単な作業ではない**
- すべての命令について，それを**実行したかどうかを監視**すればよい

  コンパイラ gcc にはそのためのオプションがあり，gcov というツールと一緒に使えば自動的にチェックできる

  来週の演習でこれを使用する

命令（めいれい）の実行（じっこう）: execution of statement
簡単（かんたん）な作業（さぎょう）ではない: it is not an easy task
自動測定（じどうそくてい）: automatic measurement
監視（かんし）する: monitor

重要

次回はホワイトボックステスト
の演習(Exercise)を行います

全員 PC を持ってくること.
そして，gcc と gcov が必要なので,
必ずインストールしておくこと.

Windows で gcc と gcov のインストールが難しい場合は,
WSL (Windows Subsystem for *Linux*) を使うとよいでしょう

Notice: all students must use your PC in the next week's exercise.
You must install gcc and gcov on your PC before the exercise.

ホワイトボックステスト法(2)
分岐網羅法

- すべての**条件分岐**(if 文, while 文, for 文)において，**True(真)となる場合と False(偽)となる場合**をそれぞれ**1回以上実行**する
  - やはり，複数の**テストケースの集合**でもって網羅できればよい
  - **全条件の「T」と「F」を経験した割合**を**分岐網羅率**という
  - これを **C1** ともいう

分岐網羅法(ぶんきもうらほう):branch coverage method
条件分岐(じょうけんぶんき):conditional branch
真(しん):true
偽(ぎ):false

分岐網羅の例：x = 1, y = 0

ソースコード	実行	T, F
void foo(int x, int y){	---	---
int sum, n;	○	---
sum = 0;	○	---
for (n = 0; n < x; n++){	○	T F
sum += n;	○	---
}	---	---
if (sum < y){	○	F
printf("%d\n", sum);	×	---
}	---	---
else{	---	---
printf("%d\n", y);	○	---
}	---	---
}	---	---

分岐網羅率(C1)

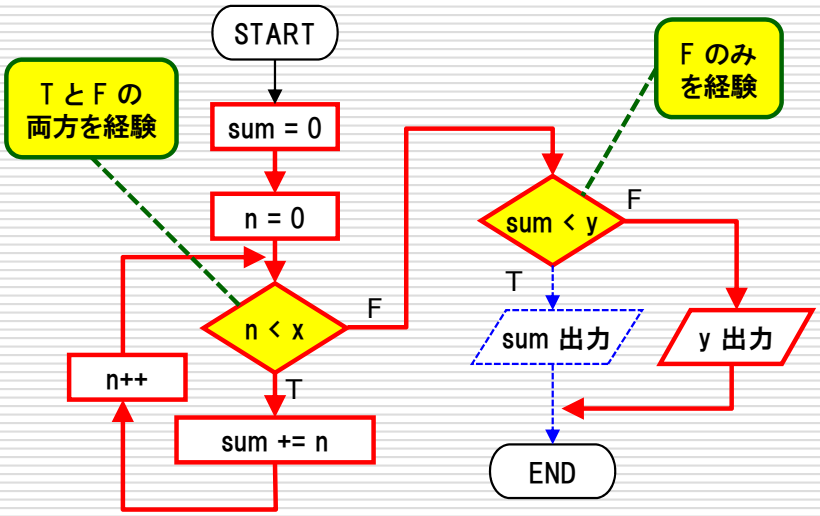
$\frac{3}{4} = 75\%$

分岐網羅率 100% の例

ソースコード	x=1, y=0	x=1, y=1	total
`void foo(int x, int y){`	---	---	---
`  int sum, n;`	---	---	---
`  sum = 0;`	---	---	---
`  for ( n = 0; n < x; n++ ){`	T F	T F	T F
`    sum += n;`	---	---	---
`  }`	---	---	---
`  if ( sum < y ){`	F	T	T F
`    printf("%d\n", sum);`	---	---	---
`  }`	---	---	---
`  else{`	---	---	---
`    printf("%d\n", y);`	---	---	---
`  }`	---	---	---
`}`	---	---	---

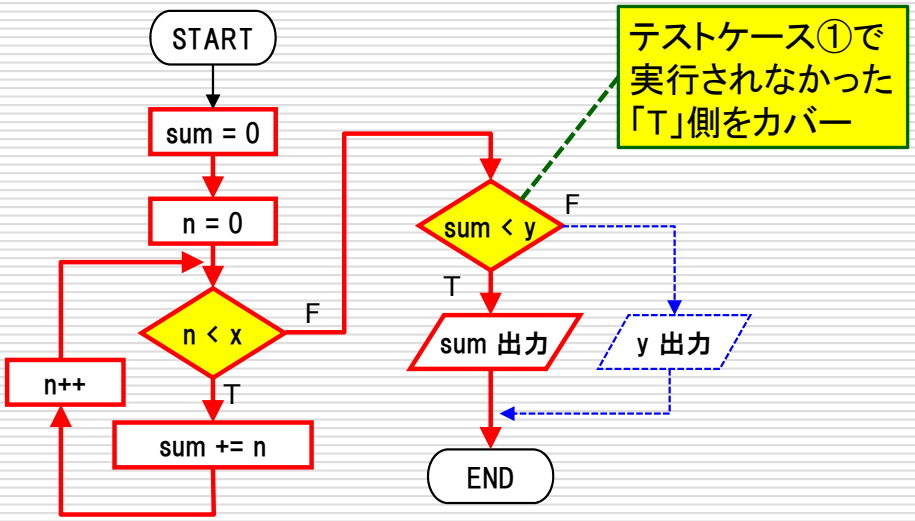
分岐網羅率（ぶんきもうらりつ）：branch coverage

フローチャートを描いた場合
テストケース① x = 1, y = 0



そのため,
分岐網羅率
(C1)は 3/4

フローチャートを描いた場合
テストケース② x = 1, y = 1



別の例でも考えてみよう

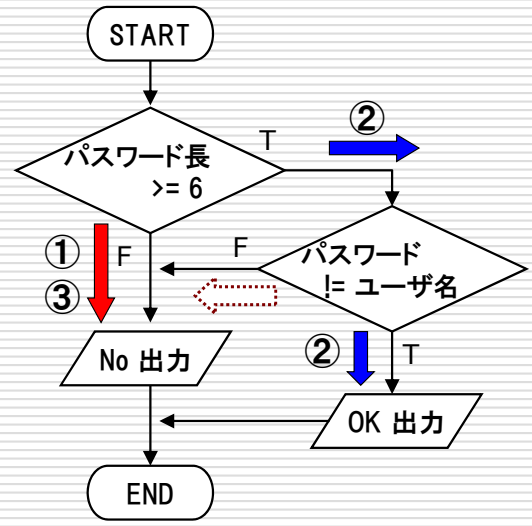
- パスワードの簡易的なセキュリティチェックを行うプログラムを考える

【チェック項目】

- パスワードの長さは6文字以上になっているか？
- パスワードがユーザ名と同じになっていないか？

簡易的な（かんいてきな）：simplified

分岐網羅率75%（=3/4）の例



テストケース

（ユーザ名，パスワード）
①（"taro", ""）
②（"taro", "foobar"）
③（"taro", "taro"）

パスワード長で6未満と6以上があり，ユーザ名とパスワードが違うものと同じものとがあるが，網羅できていない

パスワード長（ちょう）：length of password
6未満（6みまん）：less than 6
6以上（6いじょう）：greater than or equal to 6

10-minutes rest break

10分休憩

【演習1】
命令網羅率と分岐網羅率を計算せよ

（注）プログラム中の
} は命令として数えない

- 右のコードについて2つのテストケースを実行した
  - (n, limit) = (2, 1), (-2, 0)
- この場合の命令網羅率(C0)と分岐網羅率(C1)を計算せよ

```
if ( n < 0 ){
  n *= -1;
}
sum = 0;
for ( i = 0; i < n; i++ ){
  sum += i;
  if ( sum > limit ){
    sum = 0;
  }
}
printf("%d\n", sum);
```


“}” は命令（めいれい）として数えない（かぞえない）：do not count “}” as a statement

【演習2】
分岐網羅率を計算せよ

（注）return 文を実行すると，その時点で関数が終了する点に注意せよ

- 右の関数を

```
is_prime(1);
is_prime(2);
```

と呼出してテストした場合の分岐網羅率（C1）を計算せよ

```
int is_prime(int n){
  int k;
  if ( n < 2 ){
    return 0;
  }
  for ( k = 2; k < n; k++ ){
    if ( n % k == 0 ){
      return 0;
    }
  }
  return 1;
}
```


注（ちゅう） = 注意（ちゅうい）: notice, note

return 文（ぶん）を実行（じっこう）すると，その時点（じてん）で関数（かんすう）が終了（しゅうりょう）する：the function exits right after the return statement execution

関数を呼出す（よびだす）：call the function

演習2 用メモ

n=1, n=2 それぞれについて，
実行の有無を確認しながら，
条件の T, F をメモしていくとよい

ソースコード	n=1 実行	n=1 条件	n=2 実行	n=2 条件	total
`int k;`		---		---	---
`if ( n < 2 ){`					
`  return 0;`		---		---	---
`}`		---		---	---
`for ( k = 2; k < n; k++ ){`					
`  if ( n % k == 0 ){`					
`  return 0;`		---		---	---
`  }`		---		---	---
`}`		---		---	---
`return 1;`		---		---	---

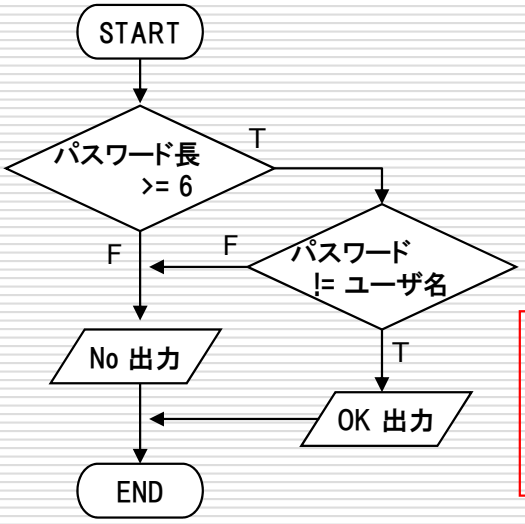
ホワイトボックステスト法(3)
条件網羅法

- **すべての条件分岐について，T・Fの組合わせを網羅**できるよう実行する
  - 例えば，2つの条件分岐があったとすると
    (F, F), (F,T), (T,F), (T,T)
    の**すべてをテスト**する
  - これらを実行できた割合を
    条件網羅率といい，
    C2 ともいう
  - ただし，**実行不可能な組合せは無視**する

最悪の場合，テストの計算量は $O(2^n)$

条件網羅法(じょうけんもうらほう):condition coverage method
T/F の組合せ(くみあわせ)を網羅(もうら):cover all combinations of True and False
実行不可能(じっこうふかのう):infeasible
無視(むし)する:omit
計算量(けいさんりょう):computational complexity

条件網羅率100%の例



組合せ		パスワード長 < 6	パスワード長 >= 6
ユーザ名とパスワード	同	ー	②
	違	①	③

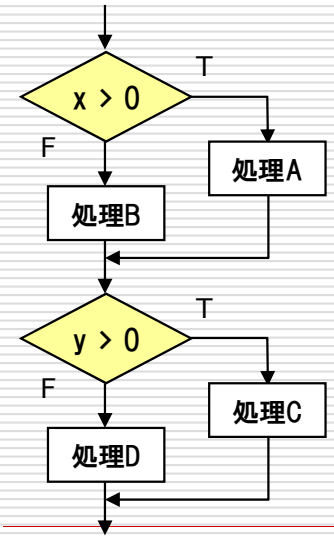
テストケース

（ユーザ名，パスワード）
① ("taro", "")
② ("momotaro", "momotaro")
③ ("taro", "foobar")

同 = 同じ（おなじ）: same

違 = 違う（ちがう）: different

(2)分岐網羅 C1 と (3)条件網羅 C2 の違い



テストケース
① x = 1, y = 1　② x = 0, y = 0

(2)分岐網羅率
= 100%

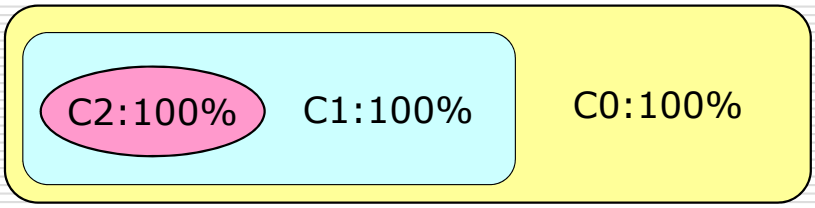
	T	F
x > 0	①	②
y > 0	①	②

(3)条件網羅率
= 50%

	x > 0	y > 0
②	F	F
	F	T
	T	F
①	T	T

三種類の包含関係

- C2:100%
  ⇒ C1:100%
- C1:100%
  ⇒ C0:100%



包含関係（ほうがんかんけい）：inclusion relation

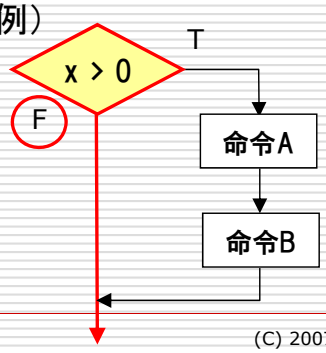
よくある間違い

- 「C1:100% ⇒ C0:100%」ということは
  「C1 ≦ C0」が成り立つのではないか？

誤り

この命題は，あくまでも網羅率100% の場合を言っているだけで他の場合については保証されない

（例）



左の条件が偽(F)の場合，
◆C1 = 1/2 = 50%
◆C0 = 1/3 ≒ 33.3%

命題(めいだい)：proposition

網羅率(もうらりつ)100%の場合(ばあい)を言っているだけ：it mentions only the case of 100% coverage

他(ほか)の場合については保証(ほしょう)されない： it gives no assurance regarding the other cases

三種類のホワイトボックステスト

- **命令網羅（C0）**は**容易に実現**できる
- **分岐網羅（C1）**はやや複雑だが，大規模でない限り**網羅率を高めることはできる**（ただし，条件分岐が多くなると厳しくなる）
- **条件網羅（C2）**は大規模・複雑になると実現可能性は**かなり厳しく**なる
  - 例えばスマートフォンの状態遷移を考えると，各画面でタップできるアイコンの個数だけ分岐があるので，組合せ総数は爆発的に増えていく

容易に（よういに）：easy
複雑（ふくざつ）：complex
大規模（だいきぼ）：large scale
厳しい（きびしい）：hard
状態遷移（じょうたいせんい）：state transition
タップ：tap
組合せ総数（くみあわせそうすう）：total number of combinations
爆発的（ばくはつてき）に増える：explode

ブラックボックステストとの関係

- ホワイトボックステストは，同値分割法といったブラックボックステストに比べると弱い
  - そもそも**要求仕様通りかどうかをテストしていない**
  - しかし，ブラックボックステストで見落とされがちな**「条件の複雑な組合せ」をチェック**できる
  - また，（仕様ではなく）**プログラムに起因する問題**があるかもしれない

ブラックボックステストを補強するという働き

要求仕様（ようきゅうしよう）通り（どおり）かどうか：whether it satisfies the requirements specification or not

見落とされがち（みおとされがち）: be likely to be overlooked, missed

プログラムに起因（きいん）する問題（もんだい）：problem caused by the program

補強する（ほきょうする）：support, reinforce

現実的にはブラックボックステストをやった上で網羅率(C0, C1)を気にする

- **仕様を満たしているか確認するのは当然必要**なので**ブラックボックステスト**によりいくつかのテストケースを実行する
- その際に**命令網羅率，分岐網羅率を気にする**ことで，ブラックボックステストで気が付かなかった**チェックの漏れ**を無くす
  - 最も基本なのは命令網羅率(C0)であり，これが100% になっていない場合，つまり，実行されない命令があるようではダメ

気にする：pay attention to
漏れ(もれ)：overlooking

テストの分類

- **ブラックボックステスト**

  プログラムの中身は見ないもの（ブラックボックス）として，**仕様に基づいた**動作テストを行う

- **ホワイトボックステスト**

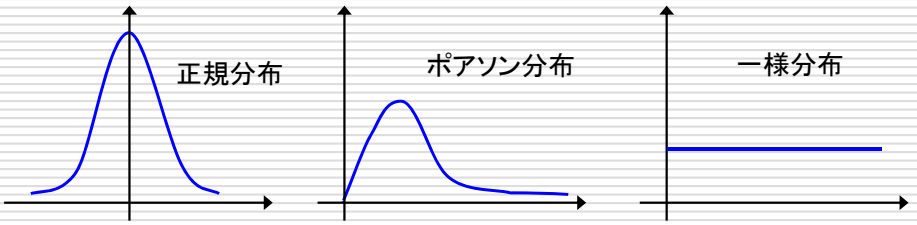
  プログラムの**内部構造（主にフローチャート）に基づいた**動作テストを行う

- **ランダムテスト**

  テストケースをランダムに（無作為に）作成して動作テストを行う

ランダムテスト法

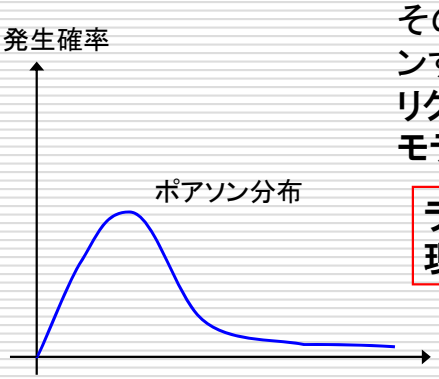
- 入力データをランダム(無作為)に作成する方法
  - どのような**確率分布**に従ってランダムデータを発生させるか，考慮すべき場合もある



ランダムテスト：random testing
確率分布（かくりつぶんぷ）：probability distribution
正規分布（せいきぶんぷ）：normal distribution
ポアソン分布（ポアソンぶんぷ）：Poisson distribution
一様分布（いちようぶんぷ）：uniform distribution

例えば，ポアソン分布を考える

- サーバへリクエストが送られてくる場合



そのような状況をシミュレーションする場合，**単位時間におけるリクエスト件数はポアソン分布でモデル化**できる

ランダムなデータとはいえ，現実に即したテストができる

サーバへリクエストが送られてくる：requests are sent to a server
単位時間（たんいじかん）：unit time
リクエスト件数（リクエストけんすう）：number of requests
現実に即した（げんじつにそくした）：realistic, close-to-reality

ランダムテスト法の利点・欠点

- 利点
  - 多数のテストケースを自動的に生成できる
  - テスト設計者，開発者が思いつかないようなテストケースが見つかることもある
- 欠点
  - 仕様に従ったテストができるわけではない
  - テストケース数の割に網羅率は低い

利点（りてん）：pros
欠点（けってん）：cons
自動的（じどうてき）に生成（せいせい）する：automatically generate
テスト設計者（テストせっけいしゃ）：test designer
開発者（かいはつしゃ）：developer
思いつかない（おもいつかない）：not come up with
テストケース数の割（わり）に網羅率（もうらりつ）は低い（ひくい）：produce a low coverage for the number of test cases

テスト法の種類と適用段階

- **ブラックボックステスト法**
  単体テストからシステムテストまで**幅広く使える**
- **ホワイトボックステスト法**
  単体テスト・(小規模な)結合テストが限界
  (システム全体の制御フローを網羅するのは難しい)
- **ランダムテスト法**
  さまざまな入力パターンを試すことで想定外のミスが無いか確認する

幅広く使える(はばひろくつかえる):broadly applicable to

システム全体(ぜんたい)の制御フロー(せいぎょフロー)を網羅(もうら)するのは難しい(むつかしい):it is hard to cover the all control flows of the whole system

さまざまな入力パターン:various input patterns

想定外(そうていがい)のミス:unexpected error

さまざまな要求のテスト

- これまで紹介した手法は主に入出力の正しさをテストするもの
- しかし実際には他にもいろんなテストが必要

いろいろな想定が必要（知識や経験も大事）

例えば，

- 性能テスト（応答時間，処理速度）
- 記憶域テスト（必要とされるメモリやハードディスクの容量）
- ストレステスト（過負荷状態でのテスト）

いろいろな想定（そうてい）：various assumption
知識（ちしき）：knowledge
経験（けいけん）：experience
性能（せいのう）：performance
応答時間（おうとうじかん）：response time
処理速度（しょりそくど）：processing speed
記憶域（きおくいき）：storage
容量（ようりょう）：volume
過負荷状態（かふかじょうたい）：overload state

まとめ

- ホワイトボックステスト：実行パスの網羅性
  - 命令網羅（C0）：　全ての命令を実行
  - 分岐網羅（C1）：　全ての分岐で T と F を実行
  - 条件網羅（C2）：　全ての分岐の組合せで T と F を実行
- ランダムテスト：　入力データをランダムに発生
- 他にも，負荷をかけて動作させるといったことも大事

実行パス（じっこうパス）：execution path

全ての命令（すべてのめいれい）を実行（じっこう）：execute all statements

全ての分岐（すべてのぶんき）で T と F を実行：execute both True and False cases for all branches

全ての分岐の組合せで T と F を実行：execute all combinations of True and False cases for all branches

負荷（ふか）をかけて動作（どうさ）させる：operate the system with putting a load on it

宿題(homework)

“[06] quiz”に答えなさい (明日の 13 時まで)

Answer “[06] quiz”
by tomorrow 13:00 (1pm)

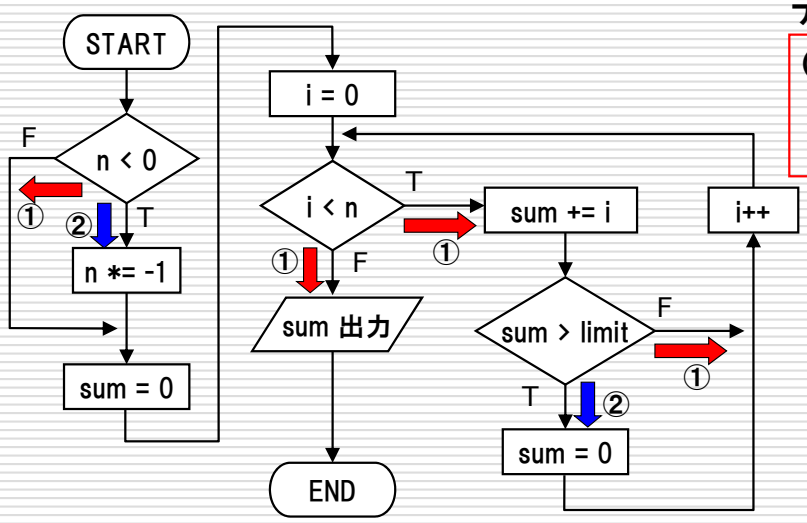
注意:quiz のスコアは final project の成績の一部となります
(Note: Your quiz score will be a part of your final project evaluation)

重要(Important Notice)

来週は演習を行います．
全員 PC が必要であり，gcc と gcov も必要です．

Next week's lecture is an exercise.
You all must do exercises on your PC.
Install gcc and gcov on your PC before the lesson.

【演習1】解答：①と②で100%



テストケース

(n, limit) =
① (2, 1)
② (-2, 0)

【演習2】解答

分岐網羅率(C1)
$\frac{3}{6} = 50\%$

ソースコード	n=1 実行	n=1 条件	n=2 実行	n=2 条件	total
int k;	○	---	○	---	---
if (n < 2){	○	T	○	F	T F
return 0;	○	---	×	---	---
}		---		---	---
for (k = 2; k < n; k++){	×		○	F	F
if (n % k == 0){	×		×		
return 0;	×	---	×	---	---
}		---		---	---
}		---		---	---
return 1;	×	---	○	---	---