Autotest.mk

ソフトウェアテスト自動化のための、GNU Make を利用したフレームワーク

j8takagi

1 Autotest.mk 概要

Autotest.mk はソフトウェアテスト自動化のための、GNU Make を利用したフレームワークです。何回か `make`コマンドを実行し、コマンドファイルとテスト結果の想定をあらわすファイルを作成すれば、テストの準備は完了です。あとは `make`コマンドを 1 回実行すれば、テストを実行できます。テストが実行されると、詳細なログが出力されます。

1.1 特徴

Autotest.mk の特徴は次のとおりです。

- 使い方がシンプル
- さまざまなプログラムや環境に対応
- ビルド作業と連携しやすい
- 無償で使うことができ、自由に改造できる

2 Autotest.mk インストール

2.1 要件

Autotest.mk は、Linux や Mac OS X などで動作します。現在動作を確認しているのは、次のディストリビューションです。

- Ubuntu Linux 10.10
- Mac OS X バージョン 10.6.5

2.2 ダウンロード

Autotest.mk のサイト (`http://www.j8takagi.net/autotest_mk/`) または github のダウンロードページ (`http://github.com/j8takagi/Autotest.mk/downloads/`) から最新版の `tar.gz`ファイルをダウンロードします。

2.3 検証

ダウンロードが完了したら、*openssl*と *diff*で正しくファイルがダウンロードができているかを検証します。

```
$ openssl md5 autotest_mk.tar.gz | diff -s autotest_mk.tar.gz.md5sum -
Files autotest_mk.tar.gz.md5sum and - are identical
```

2.4 展開

`autotest_mk`ファイルをダウンロードしたら、展開先のディレクトリーに移動してから、展開します。次のコマンドでは、現在ログインしているユーザーのホームディレクトリーに展開します。

```
$ cd ~
$ tar xvzf autotest_mk.tar.gz
```

展開したら、Autotest.mk は使えるようになります。多くのプログラムをインストールするときに必要なコンパイルやビルド、システムディレクトリーへのインストール（*./configure*や *make*、*make install*）は必要ありません。

2.5 ディレクトリー

Autotest.mk でテストを実行するときは、次の 3 つのディレクトリーを使います。

テンプレートディレクトリー
: テストグループとテストのテンプレートとなるディレクトリー

テストグループディレクトリー
: 1 つまたは複数のテストを含むディレクトリー

テストディレクトリー
: テスト実行のためのファイルと、実行結果をあらわすファイルを含むディレクトリー

3 Autotest.mk チュートリアル I - 三角形判定プログラムのテスト

概要

ここでは、Glenford J Myers『ソフトウェア・テストの技法』（近代科学社、1980）の記載を元にした、次の仕様の三角形判定プログラムをテストする手順を取り上げます[1]。

> プログラム triangle は、整数をあらわす 3 つの引数をとる。この 3 つの値は、それぞれ三角形の 3 辺の長さをあらわすものとする。プログラムは、三角形が不等辺三角形の場合は 'scalene triangle'、二等辺三角形の場合は 'isosceles triangle'、正三角形の場合は 'equilateral triangle' を印字する。

この三角形判定プログラムの動作を検証するには、次のテスト・ケースが必要です[2]。

1. 有効な不等辺三角形をあらわすテスト・ケース
2. 有効な正三角形をあらわすテスト・ケース
3. 有効な二等辺三角形をあらわすテスト・ケース
4. 有効な二等辺三角形で 2 つの等辺を含む 3 種類の組合せすべてをためすことができる 3 つのテスト・ケース
5. 1 つの辺がゼロの値をもつテスト・ケース
6. 1 つの辺が負の値をしめすテスト・ケース
7. ゼロより大きい 3 つの整数をもち、そのうち 2 つの和がそれ以外の 1 つと等しいようなテスト・ケース
8. 1 辺の長さが他の 2 辺の長さの和に等しいことを 3 種類の順列のすべてに対してためすことのできるテスト・ケース
9. ゼロより大きな 3 つの整数のうち、2 つの数の和がそれ以外の 1 つの数より小さくなるテスト・ケース
10. ゼロより大きな 3 つの整数のうち、2 つの数の和がそれ以外の 1 つの数より小さくなることにおいて、3 種類の順列すべてを考慮することのできるテスト・ケース
11. すべての辺がゼロであるテスト・ケース
12. 整数でない値をもつテスト・ケース
13. 数値の個数が間違っていることをためすテスト・ケース

三角形判定プログラムは、サンプルとして用意されているものを [準備], page 4 するか、または自作してください。

準備ができたら、Autotest.mk を使った次の手順でテストを実行できます。

1. グループディレクトリーの作成
2. テストディレクトリーの作成
3. テスト説明ファイルの作成
4. テストコマンドファイルの作成
5. テスト想定ファイルの作成
6. テストの実行
7. テストログファイルの確認
8. テストグループの実行
9. テストグループログファイルの確認

---

[1] 『ソフトウェア・テストの技法』では「カードから 3 つの整数を読む」となっているのを、「3 つの引数をとる」に変更しています。

[2] 『ソフトウェア・テストの技法』では下記の 13 個に加え、「それぞれのテスト・ケースについて、入力の値に対して予想される値をしめしたか」というテスト・ケースが記載されています。このテスト・ケースは、Autotest.mk を実行している場合は自動的に満たされると考えています。

準備

作業用ディレクトリーの作成

まず、Autotest.mk のパッケージに含まれている sample/triangle以下のファイルを、任意の作業用ディレクトリーにコピーします。ここでは、~/triangle（~は、現在ログインしているユーザーのホームディレクトリー）を作業用ディレクトリーにします。また、autotest.mkは、~に展開されているものとします。

```
$ cd
$ mkdir triangle
$ cd autotest.mk/sample/triangle
$ cp triangle.c triangle_bug.c Makefile ~/triangle
```

作業用ディレクトリーに移動し、中のファイルを確認します。

```
$ cd ~/triangle
$ ls
Makefile  triangle.c  triangle_bug.c
```

三角形判定プログラムのビルド

三角形判定プログラムをビルドします。

```
$ make
rm -f triangle
gcc -o triangle triangle.c
```

テストもかねて、三角形判定プログラムを手動で実行します。

```
$ ./triangle 3 4 5
scalene triangle
$ ./triangle 4 4 4
equilateral triangle
$ ./triangle 2 4 4
isosceles triangle
```

バグを含む三角形判定プログラムのビルド

バグを含み一部のテストに失敗する三角形判定プログラムをビルドする場合は、次のコマンドを実行します。

```
$ make bug
rm -f triangle
gcc -o triangle triangle.c
```

テストもかねて、三角形判定プログラムを手動で実行します。

```
$ ./triangle 3 4 5
futohen sankakukei
$ ./triangle 4 4 4
equilateral triangle
$ ./triangle 2 4 4
isosceles triangle
```

*./triangle 3 4 5*で、仕様で ‘scalene triangle’ となるべきところが、‘futohen sankakukei’ となっています。

3.3 グループディレクトリーの作成

テンプレートディレクトリーで *make*を実行し、グループディレクトリーを作成します。このとき、テストグループのディレクトリーを変数 GROUPDIRで指定します。ここでは、*~/triangle/test*をテストグループのディレクトリーにします。

```
$ cd ~/autotest.mk/template
$ make GROUPDIR=~/triangle/test
```

次のコマンドでグループディレクトリーに移動し、その中を確認します。

```
$ cd ~/triangle/test
$ ls
Define.mk  Makefile  Test.mk
```

3.4 テストディレクトリーの作成

グループディレクトリーで *make create*を実行し、テストディレクトリーを作成します。このとき、変数 TESTでテスト名を指定します。テスト名は、小文字のアルファベットと数字で指定します。ここでは、1 つめのテスト・ケースに対応するテストディレクトリーを、01_scaleneという名前で作成します。

```
$ make create TEST=01_scalene
```

次のコマンドでテストディレクトリーに移動し、その中を確認します。

```
$ cd 01_scalene
$ ls
Makefile
```

3.5 テスト説明ファイルの作成

テスト説明ファイル desc.txtはテストに関する説明をあらわし、テストの結果と一緒にログに出力されます。desc.txtをテキストエディターを使って次の内容で作成します。

有効な不等辺三角形をあらわすテスト・ケース

3.6 テストコマンドファイルの作成

テストコマンドファイル cmdは、テスト時に実行されるコマンドをあらわします。このコマンドにより、標準出力とエラー出力にテストの結果が出力されるようにします。

ここでは、cmdをテキストエディターを使って次の内容で作成します。

```
../../triangle 3 4 5
```

3.7 テスト想定ファイルの作成

テスト想定ファイル 0.txtは、テストが正しく実行された場合の結果をあらわします。0.txtの作成方法は、次の 3 つがあります。

- 手動で作成
- *make set*
- *make reset*

エラーが発生するテストでは、標準出力想定の下にエラー出力想定を続けます。

3.7.1 手動で作成

テスト想定ファイル 0.txtをテキストエディターを使って次の内容で作成します。

```
scalene triangle
```

3.7.2 *make set*

*make set*を実行すると、コマンドファイル cmdが実行されます。実行結果は 0.txtファイルに出力され、その内容が表示されます。

```
$ make set
scalene triangle
```

この方法で 0.txtを作成する場合は、作成された内容がテスト想定として本当に正しいかよく検討する必要があります。例えば、バグを含む三角形判定プログラムを準備して *make set*を実行すると、仕様と異なる次のようなテスト想定となり、テストを正しく実行できません。

```
$ make set
futohen sankakukei
```

0.txtがすでに存在する場合は *make set*を実行するとエラーになり、0.txtは更新されません。

3.7.3 *make reset*

0.txtがすでに存在する場合は、*make reset*で更新できます。

```
$ make reset
scalene triangle
```

3.8 テストの実行

コマンドファイルとテスト想定を用意したら、*make*または *make check*でテストを実行できます。

```
$ ls
0.txt  Makefile  cmd  desc.txt
$ make
有効な不等辺三角形をあらわすテスト・ケース
01_scalene: Test Success 2011-01-10 10:09:55
Detail in /home/foo/triangle/test/01_scalene/detail.log
```

表示される項目は、次のとおりです。

- <テスト説明ファイル desc.txtの内容>
- <テスト名>: <テスト結果（Test Success または Test Failure）> <テスト実行日時>
- Detail in <テスト詳細ログファイルの絶対パス>

バグを含む三角形判定プログラムの場合

正しいテスト想定を作成していた場合にバグを含む三角形判定プログラムでテストを実行すると、次のように表示されます。

```
$ make
make: [diff.log] Error 1 (ignored)
有効な不等辺三角形をあらわすテスト・ケース
01_scalene: Test Failure 2011-01-10 20:25:16
Detail in /home/foo/triangle/test/01_scalene/detail.log
```

3.9 テストログファイルの確認

テストを実行して作成されたファイルを確認します。1.txt summary.log detail.logが作成されています。

```
$ ls
0.txt  1.txt  Makefile  cmd  desc.txt  detail.log  summary.log
```

テストを実行すると作成されるファイルは、次のとおりです。必要に応じてファイルの内容を確認します。

1.txt　テスト結果をあらわします。エラー発生時は、標準出力のあとにエラー出力が続きます

err.log　エラー発生時に、エラーが出力されます。エラーが発生しない場合は作成されません

diff.log　テスト想定ファイル 0.txtとテスト結果ファイル 1.txtの差分をあらわします。想定と結果が同一の場合は、作成されません

summary.log
　テストの実行結果を表します。テスト実行時に表示される項目です。

detail.log
テストの詳細ログを表します。上記の内容がすべて出力されます

detail.logは、次のようになります。

```
== summary.log ==
-------------------------------------------------------------------------
有効な不等辺三角形をあらわすテスト・ケース
01_scalene: Test Success 2011-01-10 10:09:55
Detail in /home/foo/triangle/test/01_scalene/detail.log
-------------------------------------------------------------------------

== cmd ==
-------------------------------------------------------------------------
../../triangle 3 4 5
-------------------------------------------------------------------------

== 0.txt ==
-------------------------------------------------------------------------
scalene triangle
-------------------------------------------------------------------------

== 1.txt ==
-------------------------------------------------------------------------
scalene triangle
-------------------------------------------------------------------------
```

バグを含む三角形判定プログラムの場合

バグを含む三角形判定プログラムを準備してテストを実行した場合、1.txt summary.log detail.log のほかに、テスト想定とテスト結果が異なるため *diff.log*が作成されます。

```
$ ls
0.txt  1.txt  Makefile  cmd  desc.txt  detail.log  diff.log  summary.log
```

detail.logは次のようになります。

```
== summary.log ==
-------------------------------------------------------------------------
有効な不等辺三角形をあらわすテスト・ケース
01_scalene: Test Failure 2011-01-10 20:25:16
Detail in /home/foo/triangle/test/01_scalene/detail.log
-------------------------------------------------------------------------

== cmd ==
-------------------------------------------------------------------------
../../triangle 3 4 5
-------------------------------------------------------------------------

== 0.txt ==
-------------------------------------------------------------------------
scalene triangle
-------------------------------------------------------------------------

== diff.log ==
```

```
------------------------------------------------------------------------
*** 0.txt    Mon Jan 10 20:12:14 2011
--- 1.txt   Mon Jan 10 20:25:16 2011
***************
*** 1 ****
! scalene triangle
--- 1 ----
! futohen sankakukei
------------------------------------------------------------------------

== 1.txt ==
------------------------------------------------------------------------
futohen sankakukei
------------------------------------------------------------------------
```

3.9.2 テストのクリア

*make clean*を実行すると、テストの結果作成されたファイルがクリアされます。

```
$ make clean
$ ls
0.txt  Makefile  cmd  desc.txt
```

3.10 テストグループの実行

2つめ以降のテスト・ケースについても、1つめと同じ手順でテストディレクトリーの作成後、テスト説明ファイルとテストコマンドファイル、テスト想定ファイルを作成します。

ここでは、先に作成した`01_scalene`も含めて、次のようなディレクトリーとファイルを作成します。こうしたディレクトリーやファイルは Autotest.mk の`sample/triangle/test`以下にあります。

- 01_scalene
  - `Makefile desc.txt cmd 0.txt`
- 02_equilateral
  - `Makefile desc.txt cmd 0.txt`
- 03_isosceles
  - `Makefile desc.txt cmd 0.txt`
- 04_isosceles_c
  - `Makefile desc.txt cmd 0.txt`
- 05_zero
  - `Makefile desc.txt cmd 0.txt`
- 06_minus
  - `Makefile desc.txt cmd 0.txt`
- 07_line
  - `Makefile desc.txt cmd 0.txt`
- 08_lines
  - `Makefile desc.txt cmd 0.txt`
- 09_less
  - `Makefile desc.txt cmd 0.txt`
- 10_lesses
  - `Makefile desc.txt cmd 0.txt`

- 11_zeroall
  - Makefile desc.txt cmd 0.txt
- 12_notint
  - Makefile desc.txt cmd 0.txt
- 13_argcnt
  - Makefile desc.txt cmd 0.txt

テストグループディレクトリーでは、*make*または*make check*で複数のテストをまとめて実行できます。

```
$ cd ~/triangle/test
$ make
test: 13 / 13 tests passed. Detail in /home/foo/triangle/test/TEST.log
test: All tests are succeded.
```

表示されるのは、次の項目です。

- <テスト名>: <成功テスト数>/<全テスト数> test passed. Detail in <テストグループログファイルの絶対パス>
- （すべてのテストに成功した場合）<テスト名>: All tests are succeded.

バグを含む三角形判定プログラムの場合

バグを含む三角形判定プログラムを準備してテストグループを実行した場合、次のように表示されます。失敗したテストがあることをあらわしています。

```
$ cd ~/triangle/test
$ make
test: 11 / 13 tests passed. Detail in /home/foo/triangle/test/TEST.log
```

3.11 テストグループログファイルの確認

テストグループを実行すると作成されるファイルは、次のとおりです。

<テストグループ名を大文字にした文字列>.log
: テストグループに含まれるテストとその実行結果の一覧をあらわします。

Summary.log
: テストの実行結果を表します。テストグループ実行時に表示される内容です。

ここではテストグループ名はtestなので、TEST.logが次の内容で作成されます。

```
01_scalene/
有効な不等辺三角形をあらわすテスト・ケース
01_scalene: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/01_scalene/detail.log

12_notint/
整数でない値をもつテスト・ケース
12_notint: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/12_notint/detail.log

03_isosceles/
有効な二等辺三角形をあらわすテスト・ケース
03_isosceles: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/03_isosceles/detail.log

02_equilateral/
有効な正三角形をあらわすテスト・ケース
```

```
02_equilateral: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/02_equilateral/detail.log

05_zero/
1 つの辺がゼロの値をもつテスト・ケース
05_zero: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/05_zero/detail.log

13_argcnt/
数値の個数が間違っていることをためすテスト・ケース
13_argcnt: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/13_argcnt/detail.log

10_lesses/
ゼロより大きな 3 つの整数のうち、2 つの数の和がそれ以外の 1 つの数より小さくなることにおいて、3 種類の順列すべてを考慮することのできるテストケース
10_lesses: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/10_lesses/detail.log

07_line/
ゼロより大きい 3 つの整数をもち、そのうち 2 つの和がそれ以外の 1 つと等しいようなテスト・ケース
07_line: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/07_line/detail.log

06_minus/
1 つの辺が負の値をしめすテスト・ケース
06_minus: Test Success 2011-01-10 12:14:23
Detail in /home/foo/triangle/test/06_minus/detail.log

11_zeroall/
すべての辺がゼロであるテスト・ケース
11_zeroall: Test Success 2011-01-10 12:14:24
Detail in /home/foo/triangle/test/11_zeroall/detail.log

08_lines/
1 辺の長さが他の 2 辺の長さの和に等しいことを 3 種類の順列のすべてに対してためすことのできるテスト・ケース
08_lines: Test Success 2011-01-10 12:14:24
Detail in /home/foo/triangle/test/08_lines/detail.log

09_less/
ゼロより大きな 3 つの整数のうち、2 つの数の和がそれ以外の 1 つの数より小さくなるテストケース
09_less: Test Success 2011-01-10 12:14:24
Detail in /home/foo/triangle/test/09_less/detail.log

04_isosceles_c/
有効な二等辺三角形で 2 つの等辺を含む 3 種類の組合せすべてをためすことができる 3 つのテストケース
04_isosceles_c: Test Success 2011-01-10 12:14:24
Detail in /home/foo/triangle/test/04_isosceles_c/detail.log
```

テストをまとめて実行した場合も、個別のテストを実行した場合と同様に、テストディレクトリーにテストログが出力されます。特定のテスト結果を詳細に検討する場合は、そのテストのテストディレクトリーを開いてテストログファイルの確認をします。

バグを含む三角形判定プログラムの場合

バグを含む三角形判定プログラムを準備してテストグループを実行した場合、TEST.logは次のように表示されます。01_scalene のほか、11_zero のテストで失敗しています。すべての辺がゼロである場合に表示が正しくないバグがあることがわかります。

```
01_scalene/
有効な不等辺三角形をあらわすテスト・ケース
01_scalene: Test Failure 2011-01-10 21:45:52
Detail in /home/foo/triangle/test/01_scalene/detail.log

12_notint/
整数でない値をもつテスト・ケース
12_notint: Test Success 2011-01-10 21:45:52
Detail in /home/foo/triangle/test/12_notint/detail.log

03_isosceles/
有効な二等辺三角形をあらわすテスト・ケース
03_isosceles: Test Success 2011-01-10 21:45:52
Detail in /home/foo/triangle/test/03_isosceles/detail.log

02_equilateral/
有効な正三角形をあらわすテスト・ケース
02_equilateral: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/02_equilateral/detail.log

05_zero/
1 つの辺がゼロの値をもつテスト・ケース
05_zero: Test Failure 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/05_zero/detail.log

13_argcnt/
数値の個数が間違っていることをためすテスト・ケース
13_argcnt: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/13_argcnt/detail.log

10_lesses/
ゼロより大きな 3 つの整数のうち、2 つの数の和がそれ以外の 1 つの数より小さくなることにおいて、3 種類の順列すべてを考慮することのできるテストケース
10_lesses: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/10_lesses/detail.log

07_line/
ゼロより大きい 3 つの整数をもち、そのうち 2 つの和がそれ以外の 1 つと等しいようなテスト・ケース
07_line: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/07_line/detail.log

06_minus/
```

```
1 つの辺が負の値をしめすテスト・ケース
06_minus: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/06_minus/detail.log

11_zeroall/
すべての辺がゼロであるテスト・ケース
11_zeroall: Test Failure 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/11_zeroall/detail.log

08_lines/
1 辺の長さが他の 2 辺の長さの和に等しいことを 3 種類の順列のすべてに対してためすことのでき
るテスト・ケース
08_lines: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/08_lines/detail.log

09_less/
ゼロより大きな 3 つの整数のうち、2 つの数の和がそれ以外の 1 つの数より小さくなるテストケース
09_less: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/09_less/detail.log

04_isosceles_c/
有効な二等辺三角形で 2 つの等辺を含む 3 種類の組合せすべてをためすことができる 3 つのテスト
ケース
04_isosceles_c: Test Success 2011-01-10 21:45:53
Detail in /home/foo/triangle/test/04_isosceles_c/detail.log
```

3.12 ビルドとの連携

三角形判定プログラムでは、~/triangleなど作業用ディレクトリーのトップで *make*を実行すればプログラムをビルドできました。作業用ディレクトリーの Makefileを次のように編集すると、ビルドと連携してテストを実行できるようになります。

```
CC = gcc

.PHONY: all check clean

all:
$(CC) -o triangle triangle.c

check:
    $(MAKE) -c test

clean:
rm -f triangle
```

ビルドとテストは、次のように実行します。

```
$ make
gcc -o triangle triangle.c
$ make check
make -C test
test: 13 / 13 tests passed. Detail in /home/foo/triangle/test/TEST.log
test: All tests are succeded.
```

こうした設定をしておけば、プログラムを変更するたびに簡単にビルドとテストを実行することができます。

4 Autotest.mk チュートリアル II - C 言語 sort 関数のテスト

概要

ここでは C 言語の関数テストの例として、sort関数をテストします。CUnit (http://cunit.sourceforge.net/) のわかりやすい日本語チュートリアル CUnit チュートリアル (http://homepage3.nifty.com/kaku-chan/cunit/index.html) で使われている関数です。

sort_normal.cは、次の内容です。

```
void sort(int array[], int num) {
    int i;
    int j;
    int val;

    for(i=0; i<(num-1); i++) {
        for(j=(num-1); j>i; j--) {
            if (array[j-1] > array[j]) {
                val = array[j];
                array[j] = array[j-1];
                array[j-1] = val;
            }
        }
    }
}
```

バグを含む sort_bug.cは、次の内容です。

```
void sort(int array[], int num) {
    int i;
    int j;
    int val;

    for(i=0; i<(num-1); i++) {
        for(j=(num-1); j>i; j--) {
            if (array[j-1] > array[j]) {
                val = array[j];
                array[j] = array[j]; /* 本当は array[j] = array[j-1]; */
                array[j-1] = val;
            }
        }
    }
}
```

プログラムは、サンプルとして用意されているものを [準備], page 15 してください。

準備ができたら、Autotest.mk を使った次の手順でテストを実行できます。

1. グループディレクトリーの作成
2. テストディレクトリーの作成
3. テスト説明ファイルの作成
4. 関数テスト用プログラムのソース作成
5. Makefileの編集
6. テスト想定ファイルの作成
7. テストの実行

8. テストログファイルの確認
9. テストグループの実行
10. テストグループログファイルの確認

準備

作業用ディレクトリーの作成

まず、Autotest.mk のパッケージに含まれている sample/sort以下のファイルを、任意の作業用ディレクトリーにコピーします。ここでは、~/sort（~は、現在ログインしているユーザーのホームディレクトリー）を作業用ディレクトリーにします。また、autotest.mkは、~に展開されているものとします。

```
$ cd
$ mkdir sort
$ cd autotest.mk/sample/sort
$ cp sort_normal.c sort_bug.c Makefile ~/sort
```

作業用ディレクトリーに移動し、中のファイルを確認します。

```
$ cd ~/sort
$ ls
Makefile  sort_normal.c  sort_bug.c
```

sort関数ファイルの準備

*make*または *make normal*で、sort_normal.cが sort.cにコピーされます。

```
$ make
rm -f sort.c
cp sort_normal.c sort.c
```

バグを含む sort関数ファイルの準備

*make bug*で、sort_bug.cが sort.cにコピーされます。この場合、sort.cはバグを含みます。

```
$ make bug
rm -f sort.c
cp sort_bug.c sort.c
```

4.3 グループディレクトリーの作成

テンプレートディレクトリーで *make*を実行し、グループディレクトリーを作成します。このとき、テストグループのディレクトリーを変数 GROUPDIRで指定します。ここでは、*~/sort/test*をテストグループのディレクトリーにします。

```
$ cd ~/autotest.mk/template
$ make GROUPDIR=~/sort/test
```

次のコマンドでグループディレクトリーに移動し、その中を確認します。

```
$ cd ~/sort/test
$ ls
Define.mk  Makefile  Test.mk
```

4.4 テストディレクトリーの作成

グループディレクトリーで *make create*を実行し、テストディレクトリーを作成します。このとき、変数 GROUPDIRでテスト名を指定し、さらに C 言語の関数テスト用の Makefileを作成するため変数 SRC=cを指定します。ここでは、1 つめのテストを sort_001という名前で作成します [1]。

[1] 「CUnit チュートリアル」の「テスト関数 1」と「テスト関数 2」を、チュートリアルを説明しやすくするため入れ替えています。

```
$ make create TEST=sort_001 SRC=c
```

次のコマンドでテストディレクトリーに移動し、その中を確認します。

```
$ cd sort_001
$ ls
Makefile
```

4.5 関数テスト用プログラムのソース作成

C 言語の関数をテストするためのプログラムを C 言語で記述した、プログラムのソースファイル cmd.c をテキストエディターで作成します。テスト対象の関数 sortを呼び出してソートを実行し、その結果を表示するプログラムです。

```
#include <stdio.h>
#ifndef ARRAYSIZE
#define ARRAYSIZE(array) (sizeof(array)/sizeof(array[0]))
#endif

void sort(int array[], int num);

int main() {
    int i, array[] = {11, 7, 5, 3, 2};

    sort(array, ARRAYSIZE(array));
    for(i = 0; i < ARRAYSIZE(array); i++) {
        printf("%d\n", array[i]);
    }
    return 0;
}
```

4.6 Makefileの編集

Makefileをテキストエディターで編集し、テスト対象ファイルをあらわす変数 TESTTARGET_FILESを指定します。ここでは、../../sort.cを指定します。

```
include ../Define.mk
include ../Test.mk

CC := gcc
CFLAGS := -Wall

.INTERMEDIATE: $(CMD_FILE)

CMDSRC_FILE := cmd.c
TESTTARGET_FILES := ../../sort.c      # Set test target files

COMPILE_FILE := compile.log

$(CMD_FILE): $(CMDSRC_FILE) $(TESTTARGET_FILES)
$(CC) $(CFLAGS) -o $ $^ >$(COMPILE_FILE) 2>&1
cat $(COMPILE_FILE)
```

4.7 テスト説明ファイルの作成

テスト説明ファイル desc.txtはテストに関する説明をあらわし、テストの結果と一緒にログに出力されます。desc.txtをテキストエディターを使って次の内容で作成します。

```
テスト関数 1
```

4.8 テスト想定ファイルの作成

テスト想定ファイル 0.txtは、テストが正しく実行された場合の結果をあらわします。0.txtの作成方法は、次の 3 つがあります。

- 手動で作成
- *make set*
- *make reset*

エラーが発生するテストでは、標準出力想定の下にエラー出力想定を続けます。

4.8.1 手動で作成

テスト想定ファイル 0.txtをテキストエディターを使って次の内容で作成します。

```
3
```

4.8.2 *make set*

*make set*を実行すると、Makefileの設定に従い関数テスト用のプログラムソース cmd.cと関数ファイル../../sort.cからコマンドファイル cmdが作成されます。そのあと cmdが実行され、実行結果は 0.txtファイルに出力されます。実行後、次回のテストのため、cmdは自動的に削除されます。

```
$ make set
gcc -Wall -o cmd cmd.c ../../sort.c
2
3
5
7
11
rm cmd
```

この方法で 0.txtを作成する場合は、作成された内容がテスト想定として本当に正しいかよく検討する必要があります。

0.txtがすでに存在する場合は *make set*を実行するとエラーになり、0.txtは更新されません。例えば、バグを含む sort関数ファイルを準備して *make set*を実行すると、仕様と異なる次のようなテスト想定となり、テストを正しく実行できません。

```
$ make set
gcc -Wall -o cmd cmd.c ../../sort.c
2
2
2
2
2
rm cmd
```

4.8.3 *make reset*

0.txtがすでに存在する場合は、*make reset*で更新できます。

```
$ make reset
gcc -Wall -o cmd cmd.c ../../sort.c
2
```

```
3
5
7
11
rm cmd
```

4.9 テストの実行

コマンドファイルとテスト想定を用意したら、*make*または*make check*でテストを実行できます。

```
$ ls
0.txt  Makefile  cmd.c  desc.txt
$ make
sort_001: Test Success 2011-01-24 11:13:04
Details in /home/foo/sort/test/sort_001/detail.log
rm cmd
```

表示される項目は、次のとおりです。

- <テスト説明ファイル desc.txtの内容>
- <テスト名>: <テスト結果（Test Success または Test Failure）> <テスト実行日時>
- Detail in <テスト詳細ログファイルの絶対パス>

バグを含む sort関数の場合

正しいテスト想定を作成していた場合にバグを含む sort関数でテストを実行すると、次のように表示されます。

```
$ make
gcc -Wall -o cmd cmd.c ../../sort.c
make: [diff.log] Error 1 (ignored)
テスト関数 1
sort_001: Test Failure 2011-01-24 11:09:34
Details in /home/foo/sort/test/sort_001/detail.log
rm cmd
```

4.10 テストログファイルの確認

テストを実行して作成されたファイルを確認します。1.txt summary.log detail.logが作成されています。

```
$ ls
0.txt  1.txt  Makefile  cmd  desc.txt  detail.log  summary.log
```

テストを実行すると作成されるファイルは、次のとおりです。必要に応じてファイルの内容を確認します。

1.txt　テスト結果をあらわします。エラー発生時は、標準出力のあとにエラー出力が続きます

err.log　エラー発生時に、エラーが出力されます。エラーが発生しない場合は作成されません

diff.log　テスト想定ファイル 0.txtとテスト結果ファイル 1.txtの差分をあらわします。想定と結果が同一の場合は、作成されません

summary.log
　テストの実行結果を表します。テスト実行時に表示される項目です。

detail.log
　テストの詳細ログを表します。上記の内容がすべて出力されます

detail.logは、次のようになります。

```
== summary.log ==
```

```
----------------------------------------------------------------------
テスト関数 1
sort_001: Test Success 2011-01-24 11:13:04
Details in /Users/kazubito/2011_01/sort/test/sort_001/detail.log
----------------------------------------------------------------------

== cmd.c ==
----------------------------------------------------------------------
#include <stdio.h>
#ifndef ARRAYSIZE
#define ARRAYSIZE(array) (sizeof(array)/sizeof(array[0]))
#endif

void sort(int array[], int num);

int main() {
    int i, array[] = {11, 7, 5, 3, 2};

    sort(array, ARRAYSIZE(array));
    for(i = 0; i < ARRAYSIZE(array); i++) {
        printf("%d\n", array[i]);
    }
    return 0;
}
----------------------------------------------------------------------

== 0.txt ==
----------------------------------------------------------------------
2
3
5
7
11
----------------------------------------------------------------------

== 1.txt ==
----------------------------------------------------------------------
2
3
5
7
11
----------------------------------------------------------------------
```

バグを含む sort関数の場合

バグを含む sort関数を準備してテストを実行した場合、1.txt summary.log detail.logのほかに、テスト想定とテスト結果が異なるため *diff.log*が作成されます。

```
$ ls
0.txt  1.txt  Makefile  cmd  desc.txt  detail.log  diff.log  summary.log
```

detail.logは次のようになります。

```
== summary.log ==
-------------------------------------------------------------------------
テスト関数 1
sort_001: Test Failure 2011-01-24 11:09:34
Details in /home/foo/sort/test/sort_001/detail.log
-------------------------------------------------------------------------

== cmd.c ==
-------------------------------------------------------------------------
#include <stdio.h>
#ifndef ARRAYSIZE
#define ARRAYSIZE(array) (sizeof(array)/sizeof(array[0]))
#endif

void sort(int array[], int num);

int main() {
    int i, array[] = {11, 7, 5, 3, 2};

    sort(array, ARRAYSIZE(array));
    for(i = 0; i < ARRAYSIZE(array); i++) {
        printf("%d\n", array[i]);
    }
    return 0;
}
-------------------------------------------------------------------------

== 0.txt ==
-------------------------------------------------------------------------
2
3
5
7
11
-------------------------------------------------------------------------

== diff.log ==
-------------------------------------------------------------------------
*** 0.txt 2011-01-24 11:07:39.000000000 +0900
--- 1.txt   2011-01-24 11:09:34.000000000 +0900
***************
*** 1,5 ****
  2
! 3
! 5
! 7
! 11
--- 1,5 ----
  2
! 2
! 2
! 2
```

```
! 2
--------------------------------------------------------------------------

== 1.txt ==
--------------------------------------------------------------------------
2
2
2
2
2
--------------------------------------------------------------------------
```

4.10.2 テストのクリア

*make clean*を実行すると、テストの結果作成されたファイルがクリアされます。

```
$ make clean
$ ls
0.txt  Makefile  cmd.c  desc.txt
```

4.11 テストグループの実行

2 つめ以降のテスト・ケースについても、1 つめと同じ手順でテストディレクトリーの作成後、テスト説明ファイルとテストコマンドファイル、テスト想定ファイルを作成します。

ここでは、先に作成した sort_001も含めて、次のようなディレクトリーとファイルを作成します。こうしたディレクトリーやファイルは Autotest.mk の sample/sort/test以下にあります。

- sort_001
  - Makefile desc.txt cmd.c 0.txt
- sort_002
  - Makefile desc.txt cmd.c 0.txt
- sort_003
  - Makefile desc.txt cmd.c 0.txt
- sort_004
  - Makefile desc.txt cmd.c 0.txt
- sort_005
  - Makefile desc.txt cmd.c 0.txt

テストグループディレクトリーでは、*make*または*make check*で複数のテストをまとめて実行できます。

```
$ cd ~/sort/test
$ make
test:        5 / 5 tests passed. Details in /home/foo/sort/test/TEST.log
test: All tests are succeded.
```

表示されるのは、次の項目です。

- <テスト名>: <成功テスト数>/<全テスト数> test passed. Detail in <テストグループログファイルの絶対パス>
- （すべてのテストに成功した場合）<テスト名>: All tests are succeded.

バグを含む sort関数の場合

バグを含む sort関数を準備してテストグループを実行した場合、次のように表示されます。失敗したテストがあることをあらわしています。

```
$ cd ~/sort/test
```

```
$ make
test: 1 / 5 tests passed. Details in /home/foo/sort/test/TEST.log
```

4.12 テストグループログファイルの確認

テストグループを実行すると作成されるファイルは、次のとおりです。

<テストグループ名を大文字にした文字列>.log
: テストグループに含まれるテストとその実行結果の一覧をあらわします。

Summary.log
: テストの実行結果を表します。テストグループ実行時に表示される内容です。

ここではテストグループ名は testなので、TEST.logが次の内容で作成されます。

```
sort_001/
テスト関数 1
sort_001: Test Success 2011-01-24 11:37:46
Details in /home/foo/sort/test/sort_001/detail.log

sort_002/
テスト関数 1
sort_002: Test Success 2011-01-24 11:37:46
Details in /home/foo/sort/test/sort_002/detail.log

sort_003/
テスト関数 3
sort_003: Test Success 2011-01-24 11:37:46
Details in /home/foo/sort/test/sort_003/detail.log

sort_004/
テスト関数 4
sort_004: Test Success 2011-01-24 11:37:46
Details in /home/foo/sort/test/sort_004/detail.log

sort_005/
テスト関数 5
sort_005: Test Success 2011-01-24 11:37:47
Details in /home/foo/sort/test/sort_005/detail.log
```

テストをまとめて実行した場合も、個別のテストを実行した場合と同様に、テストディレクトリーにテストログが出力されます。特定のテスト結果を詳細に検討する場合は、そのテストのテストディレクトリーを開いてテストログファイルの確認をします。

バグを含む sort関数の場合

バグを含む sort関数を準備してテストグループを実行した場合、TEST.logは次のように表示されます。sort_001、sort_003、sort_004、sort_005 のテストで失敗していることがわかります。

```
sort_001/
テスト関数 1
sort_001: Test Failure 2011-01-24 12:03:02
Details in /home/foo/sort/test/sort_001/detail.log

sort_002/
テスト関数 1
```

```
sort_002: Test Success 2011-01-24 12:03:02
Details in /home/foo/sort/test/sort_002/detail.log

sort_003/
テスト関数 3
sort_003: Test Failure 2011-01-24 12:03:02
Details in /home/foo/sort/test/sort_003/detail.log

sort_004/
テスト関数 4
sort_004: Test Failure 2011-01-24 12:03:02
Details in /home/foo/sort/test/sort_004/detail.log

sort_005/
テスト関数 5
sort_005: Test Failure 2011-01-24 12:03:04
Details in /home/foo/sort/test/sort_005/detail.log
```

4.13 開発とテストの反復

あとはテスト対象となっている関数を変更するたびに、テストを実行できるようになります。つまり、関数の単位で開発とテストを反復してできます。

関数の実装だけが変更されて仕様が変わっていない場合は、テストグループのディレクトリーで *make*を実行するだけでテストができます。

関数の入力または出力の仕様が変わった場合には、0.txtと cmd.cの変更を検討します。関数のコンパイル方法（依存するファイル）が変わった場合は、Makefileの変更を検討します。

5 テンプレートディレクトリー マニュアル

テンプレートディレクトリーは、Autotest.mk をインストールするときに展開により作成された templateディレクトリーです。

5.1 *make*または*make group* - グループディレクトリーの作成

変数 GROUPDIRで指定されたディレクトリーをテストグループディレクトリーとして作成し、テストの実行に必要な次のファイルを配置します。

- Define.mk
- Makefile
- Test.mk

通常、テストに必要なファイルテンプレートディレクトリーからグループディレクトリーにコピーされます。変数 LINKMKを指定した場合は、シンボリックリンクが作成されます。

5.1.1 変数

GROUPDIR　テストディレクトリーを指定します。指定しない場合や既存のディレクトリーを指定した場合は、エラーが発生してテストグループディレクトリーは作成されません。

LINKMK　指定した場合、グループディレクトリーにテストの実行に必要なファイルのシンボリックリンクが作成されます。値は任意です。シンボリックリンクのリンク先は、テンプレートディレクトリーの中です。

使用例

グループディレクトリーとして~/triangle/testを作成します。

```
$ make GROUPDIR=~/triangle/test
```

5.2 テンプレートのファイル

テンプレートディレクトリーには、次のファイルが含まれます。

Makefile　テンプレートディレクトリーの Makefile

Group.mk　グループディレクトリーの Makefile。*make*または*make group*で、グループディレクトリーに Makefileとして配置されます。

Test.mk　テストディレクトリーから参照される Makefile。*make*または*make group*で、グループディレクトリーに配置されます。

Define.mk
　テストとテストグループに関する項目を定義した Makefile。*make*または*make group*で、グループディレクトリーに配置されます。

6 グループディレクトリー マニュアル

Section 5.1 [グループディレクトリーの作成], page 24 後、グループディレクトリーに移動すると次の操作ができます。

6.1 *make create* - テストの作成

変数 TESTで指定されたディレクトリーをテストディレクトリーとして作成し、ディレクトリーの中に Makefileを作成します。

テスト名は、小文字のアルファベットと数字で指定します。大文字ではじまる名前を指定した場合、ディレクトリーや Makefileは作成されるものの、テストグループ実行の対象になりません。

変数 SRC=cを指定した場合は、C 言語の関数テスト用の Makefileが作成されます。

6.1.1 変数

TEST　　テストディレクトリー名を指定します。

SRC　　値 cを指定すると、C 言語の関数テスト用の Makefileが作成されます。

使用例

テストディレクトリーを、01_scaleneという名前で作成します。

```
$ make create TEST=01_scalene
```

6.2 テスト名の変更とコピー

テスト名を変更したりテストをコピーしたりする場合は、*mv*や *cp*などでテストディレクトリーを移動、コピーします。

使用例

テスト名 01_scaleneを 01に変更します。

```
$ mv 01_scalene 01
```

テスト 01をコピーし、テスト 02を作成します。

```
$ cp -r 01 02
```

6.3 テストの削除

テストを削除する場合は、*rm*などでテストディレクトリーを削除します。

使用例

```
$ rm -rf 01_scalene
```

6.4 *make*または *make check* - テストグループの実行

*make*または *make check*で、グループ内にあるすべての Section 7.3 [テストの実行], page 28 をし、次の項目を表示します。

- <テスト名>: <成功テスト数>/<全テスト数> test passed. Detail in <テストグループログファイルの絶対パス>
- （すべてのテストに成功した場合）<テスト名>: All tests are succeded.

テストグループを実行すると次のファイルが作成されます。

<テストグループ名を大文字にした文字列>.log
　　テストグループに含まれるテストとその実行結果の一覧をあらわします。

Summary.log
　　テスト実行結果の概要を表します。テストグループ実行時に表示される内容です。

使用例

```
$ make
test: 13 / 13 tests passed. Detail in /home/foo/triangle/test/TEST.log
test: All tests are succeded.
```

6.5 *make checkall* - グループ内にあるすべてのテストの実行、計時

*make checkall*でグループ内にあるすべてのテストを、Section 7.3 [実行], page 28 し、Section 7.6 [計時], page 29 します。Section 6.4 [テストグループの実行], page 25 と同じ項目を表示し、出力されるファイルの種類も同じです。<テストグループ名を大文字にした文字列>.logには、テストと実行結果に加え、計時結果が一覧として出力されます。

テストの実行と計時は別に行われるため、通常にテストを実行するよりも 2 倍程度の時間がかかります。

使用例

```
$ make checkall
test: 13 / 13 tests passed. Detail in /home/foo/triangle/test/TEST.log
test: All tests are succeded.
```

6.6 *make clean* - グループの実行結果をクリア

*make clean*で、Section 6.4 [*make*または *make check*], page 25 や Section 6.5 [*make checkall*], page 26 により作成されたファイルをすべて削除します。グループディレクトリーのほか、テストディレクトリーのファイルも削除されます。

使用例

```
$ make clean
```

6.7 *make time* - グループ内にあるすべてのテストを計時

*make time*で、グループ内にあるすべてのテストを計時し、その結果が表示されます。計時結果は、<テストグループ名を大文字にした文字列>_time.logに格納されます。

使用例

```
$ make time
01_scalene/
real    0m0.007s

02_equilateral/
real    0m0.007s

03_isosceles/
real    0m0.007s

04_isosceles_c/
real    0m0.013s

05_zero/
real    0m0.010s

06_minus/
real    0m0.007s
```

```
07_line/
real    0m0.008s

08_lines/
real    0m0.024s

09_less/
real    0m0.009s

10_lesses/
real    0m0.024s

11_zeroall/
real    0m0.007s

12_notint/
real    0m0.007s

13_argcnt/
real    0m0.012s
```

6.8 *make time-clean* - グループの計時結果をクリア

*make time-clean*で、*make time*により作成されたテストグループディレクトリーと各テストディレクトリーのファイルをすべて削除します。

使用例

```
$ make time-clean
```

6.9 グループディレクトリーのファイル

グループディレクトリーには、テストディレクトリーに加えて、次のファイルが含まれます。

`Makefile` グループディレクトリーの Makefile

`Test.mk` テストディレクトリーから参照される Makefile

`Define.mk`
: テストとテストグループに関する項目を定義した Makefile

`<テストグループ名を大文字にした文字列>.log`
: Section 6.4 [*make*または*make check*], page 25 や Section 6.5 [*make checkall*], page 26 を実行したときに作成されます。*make*や*make check*の場合は、テストグループに含まれるテストと実行結果の一覧です。*make checkall*の場合は、テストと実行結果と計時結果の一覧です。

`Summary.log`
: テスト実行結果の概要です。Section 6.4 [*make*または*make check*], page 25 や Section 6.5 [*make checkall*], page 26 を実行すると作成され、その内容が表示されます。

7 テストディレクトリー マニュアル

7.1 *make set* - テスト想定の作成

*make set*で、テストコマンド cmdが実行され、テスト想定ファイル 0.txtが作成されます。また、実行時に 0.txtの内容が表示されます。0.txtがすでにある場合は、エラーが発生して 0.txtは更新されません。

使用例

```
$ make set
scalene triangle
```

7.2 *make reset* - テスト想定の作成または更新

*make reset*で、テストコマンド cmdが実行され、テスト想定ファイル 0.txtを作成されます。また、実行時に 0.txtの内容が表示されます。0.txtがすでにある場合、0.txtが更新されます。

使用例

```
$ make reset
scalene triangle
```

7.3 *make*または *make check* - テストの実行

*make*または *make check*で、テストコマンド cmdが実行され、テスト結果ファイル 1.txtが作成されます。また、実行時にテストの結果をあらわす次の項目が表示されます。

- <テスト説明ファイル desc.txtの内容>
- <テスト名>: <テスト結果（Test Success または Test Failure）> <テスト実行日時>
- Detail in <テスト詳細ログファイルの絶対パス>

使用例

```
$ make
有効な不等辺三角形をあらわすテスト・ケース
01_scalene: Test Success 2011-01-10 10:09:55
Detail in /home/foo/triangle/test/01_scalene/detail.log
```

7.4 *make clean* - テスト結果のクリア

*make clean*で、テストの実行または計時の結果作成された次のファイルが削除されます。

- 1.txt
- diff.log
- err.log
- summary.log
- time.log

使用例

```
$ make clean
$ ls
0.txt  Makefile  cmd  desc.txt
```

7.5 *make all-clean* - テストの想定と結果をクリア

*make clean*で次の、テストの想定ファイルとテストの実行または計時の結果作成されたファイルが削除されます。

- 0.txt
- 1.txt
- diff.log
- err.log
- summary.log
- time.log

使用例

```
$ make all-clean
$ ls
Makefile  cmd  desc.txt
```

7.6 *make time* - テストの計時

*make time*で、テストを計時しその結果が表示されます。計時結果は、time.logに格納されます。

使用例

```
$ make time
real  0m0.006s
```

7.7 *make time-clean* - テスト計時結果のクリア

*make time-clean*で、テストの計時の結果作成された次のファイルが削除されます。

- time.log

使用例

```
$ make time-clean
```

7.8 テストディレクトリーのファイル

Makefile テストディレクトリーの Makefile

cmd テスト時に実行されるコマンドをあらわすテストコマンドファイル。手動で作成します。

desc.txt テストに関する説明をあらわし、テストの結果と一緒にログに出力されます。手動で作成します。

0.txt テストが正しく実行された場合の結果をあらわすテスト想定ファイル。エラー発生時は、標準出力のあとにエラー出力を続けます。手動か、*make set*か、*make reset*で作成します。

1.txt テスト結果をあらわします。エラー発生時は、標準出力のあとにエラー出力が続きます。*make*または*make check*で作成されます。

diff.log テストの想定と結果の差分をあらわす差分ファイル。*make*または*make check*でテストを実行した結果、想定と差分が異なる場合に作成されます。想定と結果が同じ場合は作成されません。

err.log テストのエラー出力をあらわすエラーファイル。*make*または*make check*でテストを実行した結果、エラー出力がある場合に作成されます。なお、エラー出力は1.txtにも含まれます。

summary.log
テスト実行結果の概要です。*make*または*make check*で作成され、その内容が表示されます。

`detail.log`
: テスト実行結果の詳細です。`summary.log` `cmd 0.txt` `err.log` `diff.log` `1.txt`の内容がコピーされます。*`make`*または *`make check`*で作成されます。