ユニットテストと継続的ビルド

– コード修正の想定外の影響を早期に捕捉する –

2014 年 2 月 18 日　初稿

板垣 衛

■ 改訂履歴

稿	改訂日	改訂者	改訂内容
初稿	2014 年 2 月 18 日	板垣 衛	（初稿）

■ 目次

■ 概略

ユニットテストを作成する方法と、CI ツール（Continuous Integration : 継続的ビルド）の「Jenkins」を使用したその自動実行方法を説明する。（Jenkins 自体の説明については省略する）

安易なコード修正が思いがけず影響して別のバグを生むことがある。とくに開発末期の視野狭窄状態で起き易い問題のため、余裕をもって開発している普段から準備し続けることが重要。そのためにも、極めて手軽に記述できるユニットテストのシステムを構築する。

また、自動テスト以外にもユニットテストを効果的に利用する方法を示す。

■ 目的

本書は、ユニットテストを実践し、問題を早期発見して開発のロスを最小限にくいとどめることを目的とする。
ユニットテストは、開発中の処理を繰り返しテストする際にも役立つため、普段の開発効率を向上させることも目的として活用する。

更に、この仕組みをゲーム中の「特定の状況でのみ有効なテスト」にも活用できるものとし、QA 作業のルーチンワークに組み込んで QA 作業を効率化することも目的とする。

■ 「ユニットテスト」と「回帰テスト」

本書が実際に重視するのは、「回帰テスト」（「Regression Test : レグレッションテスト」とも呼ぶ）である。

「ユニットテスト」はその名の通り「単体テスト」のことで、特定の関数やクラスを単体で実行した時に、期待通りの結果になるかどうかをテストするものである。
「回帰テスト」は何度も繰り返し同じテストを実行することを意味する。思いがけず挙動が変わってしまったものを検出するのが目的である。

多数のユニットテストを日々回帰テストすることで、作成済みの処理の品質を保証し続けることができる。

■ ユニットテストのコーディングと実行

ユニットテストのツールはいろいろあるが、何かと融通が利くように、自作するものとする。そのため、以降の説明に記載するコードは、オリジナルのユニットテストシステムを使用したコードである。システムを自作する理由については後述する。

なお、この自作ユニットテスト処理のコーディングスタイルは Google の C++ Unit Test Framework「Google Test」を参考にしている。

▼ ユニットテストの使い方①：テスト処理の書き方

ユニットテストを実施したい処理と同一のソースファイル（.cpp）に、テスト用の処理を記述するのが基本的な使い方である。

テスト用処理は特定の書式にに従って記述する。具体的なコードのサンプルを示す。

ユニットテスト処理を記述したコードのサンプル：

【module_a.cpp】

```
//---------------------------------------
//処理モジュールＡ
class CModuleA
{
public:
	//関数１
	int func1(int p1, int p2)
	{
		return p1 + p2;
	}
	//関数２
	bool func2()
	{
		return true;
	}
	//関数３
	bool func3()
	{
		throw std::exception("Thrown message");//例外が発生
		return true;
	}
};

//---------------------------------------

#include "unit_test.h"//ユニットテスト用のインクルードファイル
//---------------------------------------
//ユニットテスト関数：モジュールＡ用
UT_BEGIN(MODULE_A, 0, UnitTest::UT_ATTR_AUTO)//他と重複しない識別名（任意の名称）を第１引数に指定する
{
	CModuleA _CModuleA;//テスト用オブジェクト（結果表示が分かり易いようにクラス名と同じような変数名にしておく）
	UT_EXPECT_EQ(_CModuleA.func1(1, 2), 3); // CModuleA::func1(1, 2) == 3 ?
	UT_EXPECT_EQ(_CModuleA.func1(3, 4), 7); // CModuleA::func1(3, 4) == 7 ?
	UT_EXPECT_NE(_CModuleA.func1(4, 5), 9); // CModuleA::func1(4, 5) != 9 ?
	UT_EXPECT_EQ(_CModuleA.func2(), true);  // CModuleA::func2() == true ?
	UT_EXPECT_NE(_CModuleA.func2(), true);  // CModuleA::func2() != true ?
```

```
	UT_EXPECT_EQ(_CModuleA.func3(), true);  // CModuleA::func3() == true ?
}
UT_END()
```

【module_b.cpp】

```
//---------------------------------------
//処理モジュールＢ
class CModuleB
{
public:
	//関数１
	int func1(int p1, int p2, int p3)
	{
		return p1 * p2 * p3;
	}
	//関数２
	void func2(int p1, int p2, int& ret)
	{
		ret = p1 * p2;
	}
};
//---------------------------------------
//処理モジュールＣ
class CModuleC
{
public:
	//関数
	float func(float p1, float p2, float& ret1, float& ret2)
	{
		ret1 = p1 + p2;
		ret2 = p1 - p2;
		return p1 * p2;
	}
};

//---------------------------------------

#include "unit_test.h"//ユニットテスト用のインクルードファイル
//---------------------------------------
//ユニットテスト関数：モジュールＢ用
UT_BEGIN(MODULE_B, 0, UnitTest::UT_ATTR_AUTO)//他と重複しない識別名（任意の名称）を第１引数に指定する
{
	CModuleB _CModuleB;//テスト用オブジェクト（結果表示が分かり易いようにクラス名と同じような変数名にしておく）
	int out = 0;
	UT_EXPECT_EQ(_CModuleB.func1(1, 2, 3), 1+2+3); // CModuleB::func1(1,  2,  3) == 1 + 2 + 3 ?
	UT_EXPECT_NE(_CModuleB.func1(3, 4, 5), 60);    // CModuleB::func1(3,  4,  5) != 60 ?
	UT_EXPECT_GT(_CModuleB.func1(5, 6, 7), 10);    // CModuleB::func1(5,  6,  7) >  10 ?
	UT_EXPECT_GE(_CModuleB.func1(7, 8, 9), 20);    // CModuleB::func1(7,  8,  9) >= 20 ?
	UT_EXPECT_LT(_CModuleB.func1(9, 10, 11), 30);  // CModuleB::func1(9,  10, 11) <  30 ?
	UT_EXPECT_LE(_CModuleB.func1(11, 12, 13), 40); // CModuleB::func1(11, 12, 13) <= 40 ?
	UT_EXPR(_CModuleB.func2(1, 2, out)); UT_EXPECT_EQ_CHILD(out, 2);  // CModuleB::func2(1, 2, out); out == 2 ?
	UT_EXPR(_CModuleB.func2(3, 4, out)); UT_EXPECT_NE_CHILD(out, 12); // CModuleB::func2(3, 4, out); out != 12 ?
}
UT_END()
//---------------------------------------
//ユニットテスト関数：モジュールＣ用（無理に分ける必要はないが分けても書ける）
UT_BEGIN(MODULE_C, 0, UnitTest::UT_ATTR_NOAUTO)//他と重複しない識別名（任意の名称）を第１引数に指定する
{
	CModuleC _CModuleC;//テスト用オブジェクト（結果表示が分かり易いようにクラス名と同じような変数名にしておく）
	float out1 = 0.f;
	float out2 = 0.f;
	UT_EXPR_WITH_RET(_CModuleC.func(1.1f, 2.2f, out1, out2));
	                                    UT_EXPECT_EQ_CHILD(out1, 2.f);  UT_EXPECT_LT_CHILD(out2, 0.f);
	                                    // CModuleC::func(1.1f, 2.2f, out1, out2); out1 ==  2.f ? out2 <   0.f ?
	UT_EXPR_WITH_RET(_CModuleC.func(3.3f, 4.4f, out1, out2));
```

```
                                    UT_EXPECT_NE_CHILD(out1, 7.f);  UT_EXPECT_GT_CHILD(out2, 0.f);
                                    // CModuleC::func(3.3f, 4.4f, out1, out2); out1 !=  7.f ? out2 >   0.f ?
    UT_EXPR_WITH_RET(_CModuleC.func(5.5f, 6.6f, out1, out2));
                                    UT_EXPECT_LE_CHILD(out1, 20.f); UT_EXPECT_GE_CHILD(out2, 10.f);
                                    // CModuleC::func(5.5f, 6.6f, out1, out2); out1 <= 20.f ? out2 >= 10.f ?
    UT_EXPR_WITH_RET(_CModuleC.func(7.7f, 8.8f, out1, out2));
                                    UT_EXPECT_LT_CHILD(out1, 1.f);  UT_EXPECT_LE_CHILD(out2, 2.f);
                                    // CModuleC::func(7.7f, 8.8f, out1, out2); out1 <   1.f ? out2 <=  2.f ?
}
UT_END()
```

「UT_BEGIN()」マクロ～「UT_END()」マクロの範囲がテスト用関数となる。

多数のテスト用関数が作られても競合しないように、「UT_BEGIN()」の第一引数に任意の識別名を渡す。大抵はクラス名をそのまま指定する。

この関数の中で「UT_EXPECT_***()」マクロを使用し、計算式と期待する結果を書くことでテストする。

関数の戻り値を評価したくない、あるいは、戻り値がない関数を扱いたい場合にも対応する。「UT_EXPR_***()」マクロは関数の実行だけを行い、結果を評価せず、ログ出力だけを行う。このような関数の処理の結果としては、参照渡しの引数やグローバル変数に返された値を評価する。「UT_EXPECT_**_CHILD()」マクロを使用することで、そのような値の評価を行える。

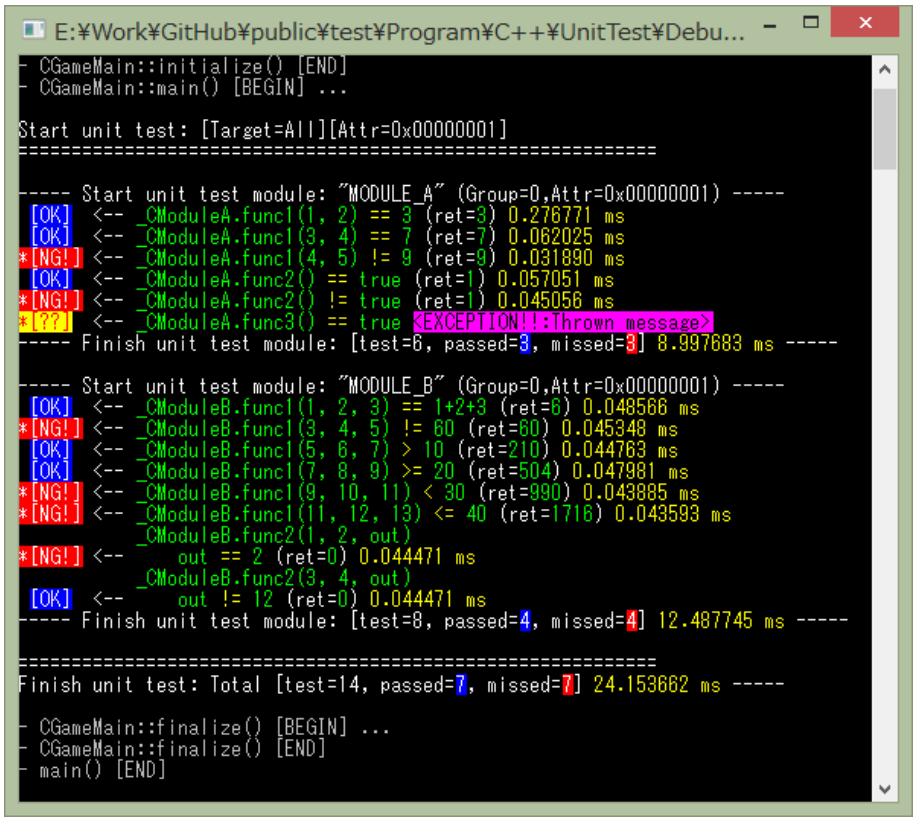
重要なポイントとして、記述すべきテストは「正常動作のパターン」だけではなく「エラー値が返るパターン」も検証すべきである。

期待通りのエラーが返るなら「テストとしてはOK」ということになる。期待通りに処理が失敗することもまた重要なテストである。

▼ ユニットテストの使い方②：実行結果

ユニットテストを実行すると、下記のような結果が出力される。

ユニットテスト実行結果のサンプル：



このサンプルでは「OK だったもの」「NG だったもの」「例外が発生して結果が得られなかったもの」が示されている。（例外は NG にカウントされる）

ユニットテストでは、1 件の NG も許されない。なぜなら、NG になるようなテストは一件も登録していないはずだからである。

これは、「関数がエラー終了した」という意味ではなく、正常終了にしろエラー終了にしろ、「期待通りの結果ではない」という意味である。1 件でもそのような結果が返ったら、最優先で修正しなければならない。

なお、テスト結果には処理時間も表示されるため、パフォーマンスのチェックを兼ねることができる。

また、テスト結果にサンプルで用意した「MODULE_C」が表示されていないのは、「MODULE_C」は自動実行しない設定で登録していたからである。

例えば、「特定のあるステージでしかテストできない」といった特殊な条件が必要なテストは自動実行しない設定でテストを登録することができる。

▼ ユニットテストの使い方③：実行の指定

先に実行結果を示したが、実際にはテスト処理を書いただけではまだ実行できない。

テストを実行する処理を記述する必要がある。

一般的なユニットテストは、main() 関数の最初に実行して即終了することが多いが、ゲームの場合、「メモリマネージャが初期化された後でないとまともに動作する処理が少ない」といった事情があるため、「初期化処理の後」などの任意のタイミングで呼び出すようにする。

ユニットテスト処理を実行するコードのサンプル：

【main.cpp】

```
#include "unit_test.h"//ユニットテスト用のインクルードファイル

//メイン処理
int main(const int argc, const char* argv[])
{
	//初期化処理
	//…（略）…

	//ユニットテスト実行
	UT_RUN_MAIN();//ユニットテスト実行モードで実行されたら、この処理の実行後に即 return する。エラー数を返す。

	//…（略）…
}
```

ユニットテストを実行したいタイミングで「UT_RUN_MAIN()」マクロを実行する。

このマクロは、ユニットテストの実行が終了すると return するので注意。エラー数を返すため、Windows なら環境変数「%ERRORLEVEL%」をチェックすることでエラーの有無がわかる。

● 手動実行

なお、手動実行したいテストは別途記述することができる。ゲーム上の特定の状況で実行するためのテストなどを登録し、デバッグメニューから実行できるようになどして使用する。

ユニットテスト処理を手動実行するコードのサンプル：

【debug_unit_test.cpp】

```
#include "unit_test.h"//ユニットテスト用のインクルードファイル

//デバッグ処理　※デバッグメニューから実行される
void debugUnitTest(DEBUG_DESC desc, const int argc, const char* argv[])
{
	//ユニットテスト手動実行：モジュールC
	UT_OUTPUT("手動ユニットテスト:MODULE(¥"MODULE_C¥")¥n");//ユニットテスト有効時のみ処理されるプリント文
	UT_RUN_MODULE("MODULE_C", UnitTest::UT_ATTR_ANY);//モジュールを指定して実行
}
```

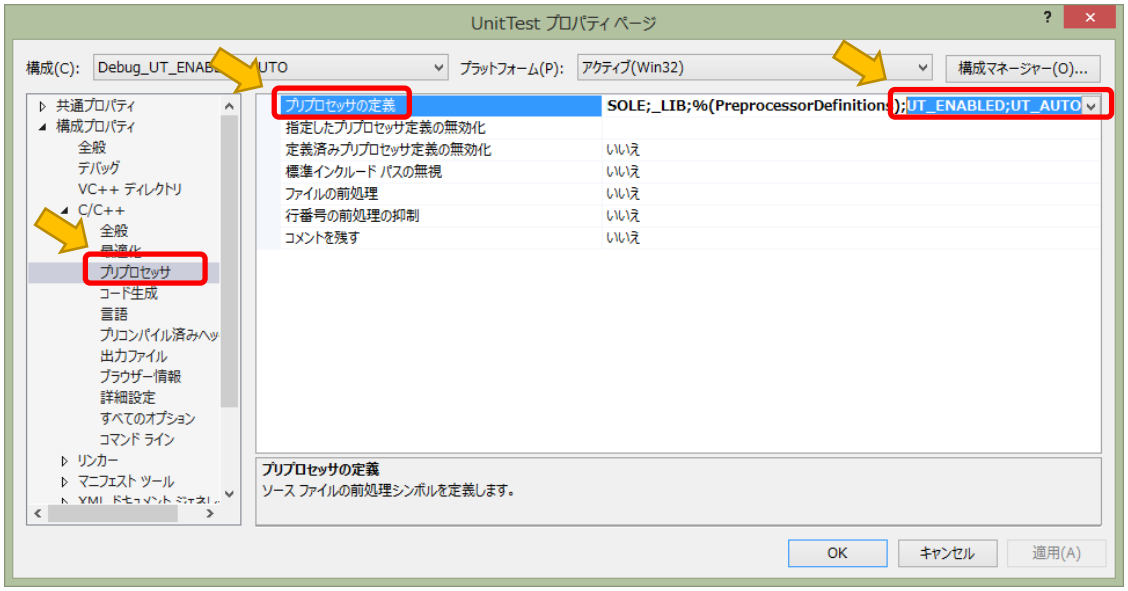
▼ ユニットテストの使い方④：ビルド方法

ユニットテスト処理は、ゲーム用プロジェクトにそのまま記述する。

しかし、通常のビルドではユニットテストは無効となり、普通にゲームが起動する。この時、ユニットテストの全ての処理コードは、実行時のプログラムから完全に消滅している。

ユニットテストを有効にするには、特定のマクロをプリプロセッサに指定する。

ユニットテスト処理を有効化するビルド設定のサンプル：



マクロ「UT_ENABLED」でユニットテストの処理コードを有効化し、マクロ「UT_AUTO」で「UT_RUN_MAIN()」マクロによる実行を許可する。

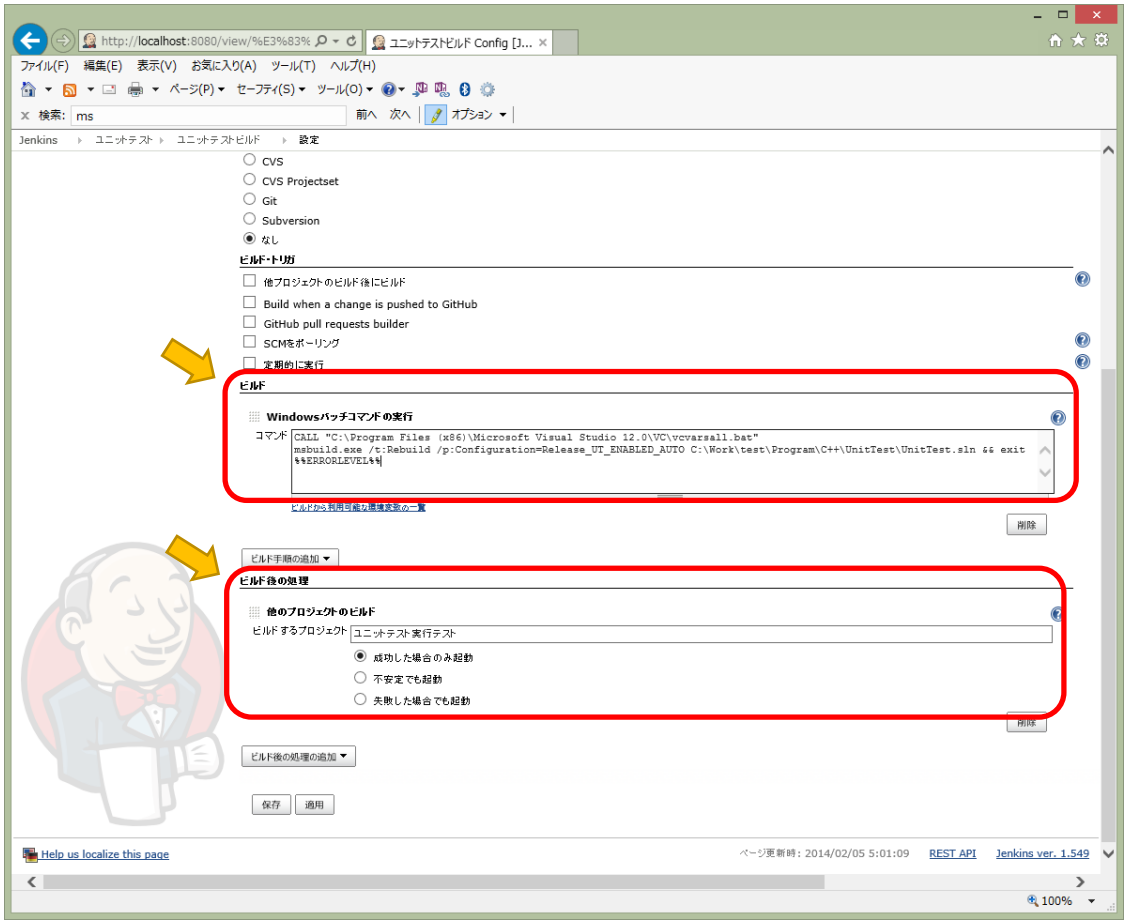
このように、マクロの設定によってユニットテストの有効化／無効化を切り替えできるため、専用プロジェクトを作成する必要はない。ただし、CI ツールで自動ビルドと自動実行をできるように、専用のビルド構成は用意したほうが良い。

▼ ユニットテストの使い方⑤：Jenkins の設定

Jenkins から自動的にユニットテストを実行する場合は、最低限ユニットテスト構成のビルドとその実行ができればよい。(Jenkins の使い方そのものは解説しない)

Jenkins のジョブ設定サンプル：

Jenkins のビルド設定サンプル：



これは、Visual Studio 2013 のソリューションに対して、MSBuild.exe を使用してビルドするサンプルである。普段 Visual Studio のパスを通していないことや、ビルド対象の構

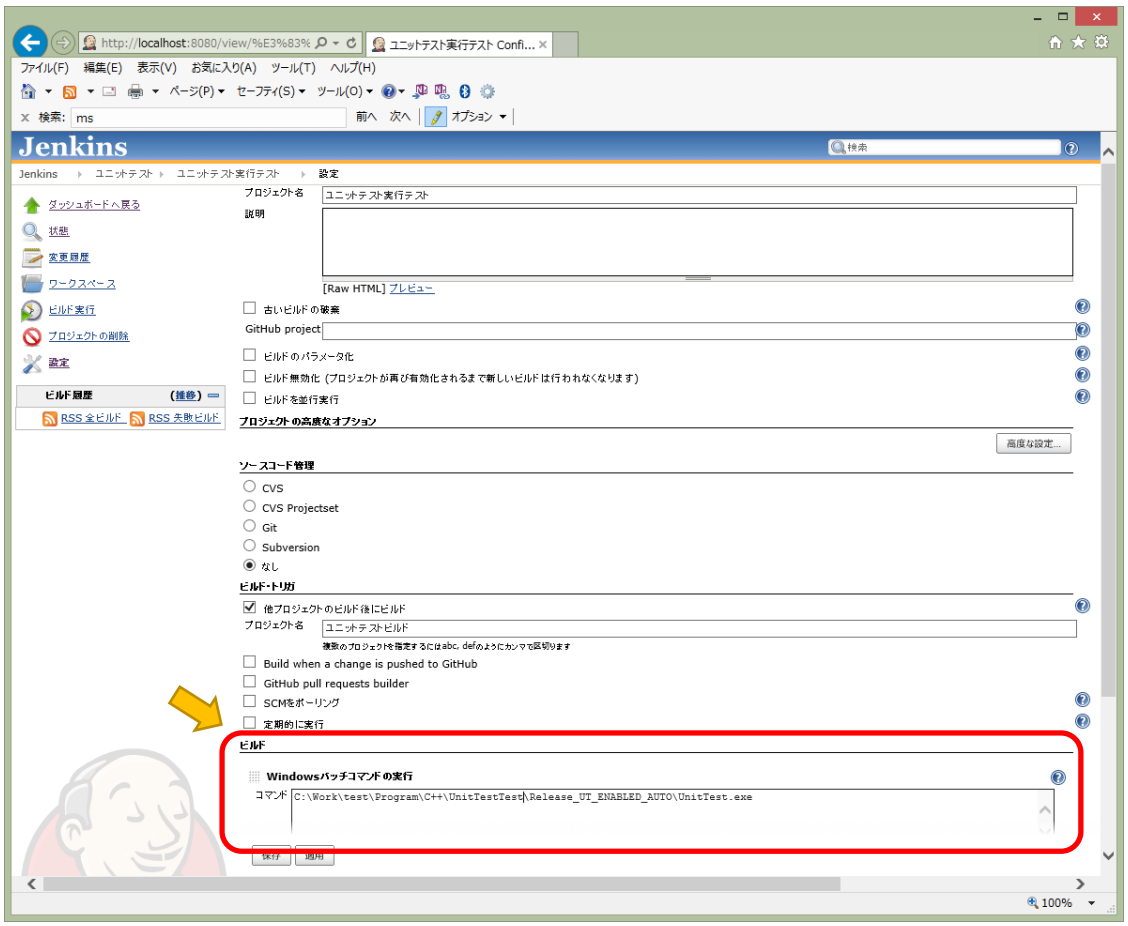
成とターゲット（Rebuild）を指定したかったことから、MSBuild プラグインは使用せず、バッチコマンドを使用している。ビルド成功後は、そのままユニットテストを実行するようにジョブ（プロジェクト）を連結している。

　また、ここでは使っていないが、「SCM をポーリング」の設定と組み合わせて、定期ビルドを待たず、コミットがある度にテストすることも推奨する。

Jenkins からバッチコマンドで Visual Studio ソリューションをビルドするサンプル：

```
rem 環境変数を設定し、MSBuildexe が実行できるようにパスを通す
CALL "C:¥Program Files (x86)¥Microsoft Visual Studio 12.0¥VC¥vcvarsall.bat"
rem MSBuild.exe を使用してビルド：構成「Release_UT_ENABLED_AUTO」をリビルド
msbuild.exe  /t:Rebuild  /p:Configuration=Release_UT_ENABLED_AUTO  C:¥Work¥test¥Program¥C++¥UnitTestTest¥Test.sln  &&
exit %%ERRORLEVEL%%
```

Jenkins のユニットテスト実行設定サンプル：



　ユニットテスト用にビルドしたプログラムをそのまま実行しているだけである。

　他、失敗時のメール通知の設定なども行うべきだが、割愛する。

Jenkins からバッチコマンドでユニットテストを実行するサンプル：

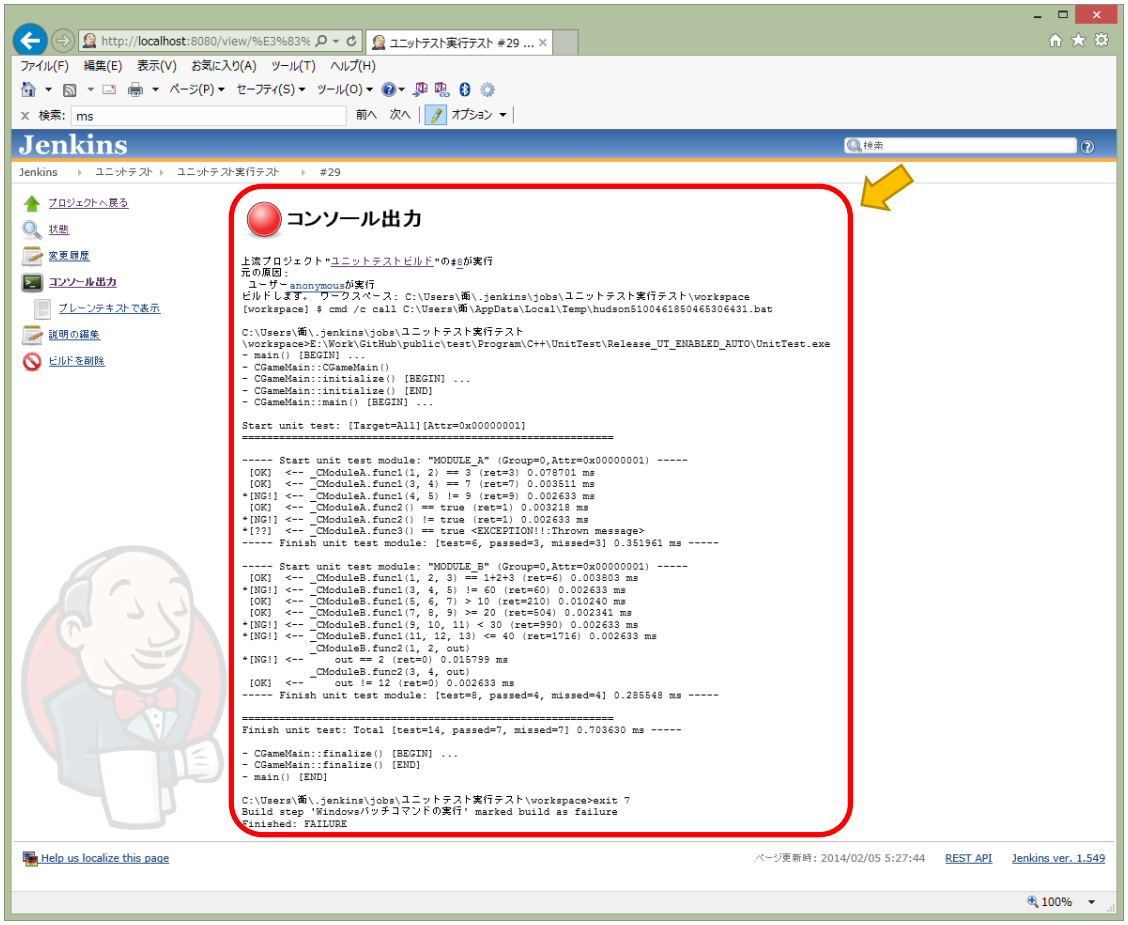
```
rem ユニットテストを実行
C:¥Work¥test¥Program¥C++¥UnitTestTest¥Release_UT_ENABLED_AUTO¥UnitTest.exe
```

▼ ユニットテストの使い方⑥：Jenkins の実行結果

Jenkins からのユニットテストの実行結果のサンプルを示す。

Jenkins のビルド～ユニットテスト実行結果サンプル：※エラーが出ている状態

Jenkins のユニットテスト実行結果（コンソール出力）サンプル：※エラーが出ている状態



▼ ユニットテストシステムを自作するメリット

ここまでのサンプルで示したように、ユニットテストシステムを自作することで、「「自動実行」と「手動実行」を切り分けて状況に合わせて実行する」といった仕組みを拡張し易くする。「ゲーム開発」にフォーカスして、より使い易いシステムを構築する。

■ ユニットテストの応用

▼ コミットごとの自動回帰テスト

ここまで説明してきたように、ユニットテストは回帰テストとして定期実行することで効果がある。

Jenkins の説明にも記述したが、Jenkins にはコミットを監視してジョブを実行する仕組みがあるので、コミットがあるたびに回帰テストを実行するのも望ましい。

「回帰テストの失敗」は、最優先で対処すべき問題である。この問題の早期発見は、開発時のロスを最小限にくいとどめることになる。

▼ テストファースト

「テストファースト」（Test First）もしくは「テスト駆動開発」（TDD = Test Driven Development）という考え方がある。

これは、処理の実装に先駆けてテストコードを書くことにより、道を踏み外さずに確かなコーディングを行う手法である。コードの品質を高めるために有効。

このような開発を実践するにあたっても、ユニットテストのシステムを便利に使用できる。

● ペア・プログラミング

XP（eXtreme Programing：エクストリーム・プログラミング）というアジャイル開発方法論の中には「ペア・プログロミング」というプラクティスがある。

これは、その名の通り「ペア」でプログラミングを行う事である。

二人組でいっしょにコーディングを行うプラクティスで、時間などで区切って一人がコーディングし、一人がそれを見ながら指摘や相談相手となる。

その目的は、誤ったコーディングにならないように注意力を高めることや、重要なコードの共有を行うことであるが、経験の浅いスタッフに「指導」するために実践するケースもある。この時、指導者側が先に「テストコード」を書き、正しい結果になるように生徒側がコーディングするといった手法も用いられる。

▼ QA のルーチンワーク

ゲームをビルドし、制作スタッフや QA スタッフに配布する際には、最低限の動作チェックを行うものであるが、ユニットテストの仕組みはそのチェック作業を効率化する。

また、特定の状況で手動実行するユニットテストも多数用意しておくことで、QA 作業の効率化も期待できる。

QA チームに渡されたビルド（もしくはデイリービルド）のチェックに入る際に、手動ユニットテストを一通り実行することをルーチンワークとすることで、それが QA チェックに値するビルドであるかどうかの判断に役立てることができる。

■ 処理実装サンプル

今回のサンプルのために作成したユニットテストシステムのプログラムを（やや乱暴だが）全文記載する。

コメントが少なめで少々分かりにくいが、そのままの状態とする。あくまでも参考用である。

まだまだ改善の余地がある状態で、例えば、大量のテストコードが実装された時の対処を組み込んでいない。

ユニットテストを書く習慣が定着すると、膨大な量のテストコードが追加されることになる。自動実行はそれで良いが、手動実行したい時は普通にゲームが起動する必要があるものの、プログラムサイズがメモリを圧迫して起動できなくなる可能性がある。この対処として、特定のテスト処理だけ実体化されるような工夫が必要。

ほか、new / delete や std::string を使用している点も改善のポイント。

▼ ユニットテストシステムサンプル

ユニットテストシステムのサンプルを示す。赤字はユニットテスト処理作成時に直接使用するマクロ／関数。

【unit_test.h】

```
#pragma once
#ifndef __UNIT_TEST_H__
#define __UNIT_TEST_H__

//プロジェクト設定の［C/C++］→［プリプロセッサ］→［プリプロセッサの定義］にて定義されるマクロ
//#define UT_ENABLED       ... ユニットテストの処理コードが有効になる。
//
//#define UT_AUTO          ... ユニットテストの自動実行を有効にする。
//                             ※main 処理の冒頭で「UT_RUN_MAIN();」と記述しておく。
//                               以後の処理は実行せずに return する。この return 値を main 関数の戻り値にするように。
//                               メモリ管理などの基本的な初期化処理が終わった後で実行するように。
//#define UT_TARGET_MODULE ... UT_AUTO で自動実行時に、指定のモジュール名を対象に限定して実行する。
//                             省略した場合は全て対象。
//#define UT_TARGET_GROUP  ... UT_AUTO で自動実行時に、指定のグループ ID を対象に限定して実行する。
//                             省略した場合は全て対象。
//
//※これらのマクロを使用せずとも、「UT_RUN_ALL();」「UT_RUN_MODULE();」「UT_RUN_GROUP();」のいずれかを直接呼び出せば、
//  いつでもユニットテストを実行可能。
//  戻り値はテストにミスした数。これは常に 0 でなければならない。

#ifdef UT_ENABLED

#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <string>
#include <sstream>
#include <exception>
#include <assert.h>
```

```
#ifndef ASSERT
#define ASSERT(expr) assert(expr)
#endif//ASSERT

namespace UnitTest
{
	class CCollection;
	class CElapsedTime;

	//ユニットテスト実行属性
	typedef unsigned int T_UT_ATTR;
	enum E_UT_ATTR
	{
		UT_ATTR_NONE = 0x00000000, //属性無し
		UT_ATTR_ANY = 0xffffffff,  //全て
		UT_ATTR_AUTO = 0x00000001, //自動実行用
		UT_ATTR_NOAUTO = 0x00000002, //非自動実行用
	};

	//ユニットテスト比較演算子指定用定数
	enum E_UT_OPE
	{
		UT_OPE_UNKNOWN = 0, //??
		UT_OPE_EQ, //==
		UT_OPE_NE, //!=
		UT_OPE_GT, //>
		UT_OPE_GE, //>=
		UT_OPE_LT, //<
		UT_OPE_LE, //<=
		UT_OPE_NUM,
	};
	struct T_UT_OPE_UNKNOWN_DUMMY{};
	struct T_UT_OPE_EQ_DUMMY{};
	struct T_UT_OPE_NE_DUMMY{};
	struct T_UT_OPE_GT_DUMMY{};
	struct T_UT_OPE_GE_DUMMY{};
	struct T_UT_OPE_LT_DUMMY{};
	struct T_UT_OPE_LE_DUMMY{};

	//ユニットテストメインクラス
	class CCollection
	{
	private:
	//	explicit CUnitTestMain(){}
		explicit CCollection(CCollection&){}
	public:
		typedef int(*UNIT_TEST_FUNC_P)(int& passed, int& missed);
		typedef int(*UNIT_TEST_OUTPUT_FUNC_P)(const char* fmt, va_list list);
		struct UNIT_TEST_FUNC_INFO
		{
			CCollection::UNIT_TEST_FUNC_P m_func;
			const char* m_moduleName;
			int m_groupId;
			T_UT_ATTR m_attr;
			int passed;
			int missed;
			double elapsed_time;
		};
	public:
		//コンストラクタ
		explicit CCollection(){}
		//デストラクタ
		~CCollection(){}

	public:
```

```
        //アクセッサ
        static int getFuncInfoListNum();
        static const UNIT_TEST_FUNC_INFO* getFuncInfoListTop();
        static const UNIT_TEST_FUNC_INFO* getFuncInfo(const int index);
        static const UNIT_TEST_FUNC_INFO* getFuncInfo(const char* module_name);

        //前回の結果
        static int getLastPassedTotal();
        static int getLastMissedTotal();

    public:
        //前回の実行結果をリセット
        static void resetLastResult();

    public:
        //ユニットテスト実行
        static int runUnitTest(const char* target_module_name = nullptr, const int target_group_id = 0,
                                                                const T_UT_ATTR target_attr = UT_ATTR_ANY);
        static int runUnitTestStandard(const T_UT_ATTR target_attr = UT_ATTR_ANY);

    public:
        //ユニットテスト登録
        static bool addFuncInfo(UNIT_TEST_FUNC_P func, const char* module_name, const int group_id,
                                                                                    const T_UT_ATTR attr);

        //ユニットテスト結果表示
        static void setOutputFunc(UNIT_TEST_OUTPUT_FUNC_P func);
        static int output(const char* fmt, ...);
        static void outputRunUTBegin(const char* target_module_name, const int target_group_id,
                                                                        const T_UT_ATTR target_attr);
        static void outputRunUTEnd(const char* target_module_name, const int target_group_id,
                                                        const T_UT_ATTR target_attr, const int total_passed,
                                                        const int total_missed, const double elapsed_time);
        static void outputRunUTModuleBegin(const char* module_name, const int group_id, const T_UT_ATTR attr);
        static void outputRunUTModuleEnd(const char* module_name, const int group_id, const T_UT_ATTR attr,
                                                const int passed, const int missed, const double elapsed_time);

        //ユニットテスト結果格納クラス
        class CExprCResultObjBase
        {
        public:
            CExprCResultObjBase();
            ~CExprCResultObjBase();

        public:
            void setResult(const bool result){ this->m_result = result; }
            bool getResult() const { return this->m_result; }
            void setHasResult(const bool has_result){ this->m_hasResult = has_result; }
            bool hasResult() const { return this->m_hasResult; }
            void setException(std::exception const& e);
            bool hasException() const { return this->m_hasException; }
            bool hasExprStr() const { return this->m_hasExprStr; }
            bool hasValueStr() const { return this->m_hasValueStr; }
            bool hasOpeStr() const { return this->m_hasOpeStr; }
            bool hasExpectStr() const { return this->m_hasExpectStr; }
            void setExprStr(const char* expr);
            const char* getExprStr() const { return m_exprStr.c_str(); }
            void setValueStr(const char* value);
            const char* getValueStr() const { return m_valuieStr.c_str(); }
            void setOpeStr(const char* ope);
            const char* setOpeStrFromId(const E_UT_OPE ope);
            const char* getOpeStr() const { return this->m_opeStr.c_str(); }
            void setExpectStr(const char* expect);
            const char* getExpectStr() const { return this->m_expectStr.c_str(); }
            const char* getExceptionStr() const { return this->m_exceptionStr.c_str(); }
```

```
private:
	bool m_result;
	bool m_hasResult;
	bool m_hasException;
	bool m_hasExprStr;
	bool m_hasValueStr;
	bool m_hasOpeStr;
	bool m_hasExpectStr;
	std::string m_exprStr;
	std::string m_valuieStr;
	std::string m_opeStr;
	std::string m_expectStr;
	std::string m_exceptionStr;
};
template<typename T>
class CExprCResultObj : public CExprCResultObjBase
{
public:
	CExprCResultObj(const T value, const T expect) :
		m_hasValue(true),
		m_hasExpect(true),
		m_value(value),
		m_expect(expect),
		CExprCResultObjBase()
	{
	}
	CExprCResultObj(const T value) :
		m_hasValue(false),
		m_hasExpect(false),
		m_value(0),
		m_expect(0), CExprCResultObjBase()
	{
		this->setValue(value);
	}
	CExprCResultObj() :
		m_hasValue(false),
		m_hasExpect(false),
		m_value(0),
		m_expect(0),
		CExprCResultObjBase()
	{
	}
	~CExprCResultObj()
	{
		this->~CExprCResultObjBase();
	}

public:
	void setValue(const T value)
	{
		this->m_value = value;
		this->m_hasValue = true;
		std::ostringstream os;
		os << value;
		this->setValueStr(os.str().c_str());
	}
	T getValue() const { return this->m_value; }
	void setExpect(const T expect)
	{
		this->m_expect = expect;
		this->m_hasExpect = true;
	}
	T getExpect() const { return this->m_expectStr; }
```

```
public:
	void setOpeStrFromType(T_UT_OPE_UNKNOWN_DUMMY ope)
	{
		this->setOpeStrFromId(UT_OPE_UNKNOWN);
	}
	void setOpeStrFromType(T_UT_OPE_EQ_DUMMY ope)
	{
		this->setOpeStrFromId(UT_OPE_EQ);
	}
	void setOpeStrFromType(T_UT_OPE_NE_DUMMY ope)
	{
		this->setOpeStrFromId(UT_OPE_NE);
	}
	void setOpeStrFromType(T_UT_OPE_GT_DUMMY ope)
	{
		this->setOpeStrFromId(UT_OPE_GT);
	}
	void setOpeStrFromType(T_UT_OPE_GE_DUMMY ope)
	{
		this->setOpeStrFromId(UT_OPE_GE);
	}
	void setOpeStrFromType(T_UT_OPE_LT_DUMMY ope)
	{
		this->setOpeStrFromId(UT_OPE_LT);
	}
	void setOpeStrFromType(T_UT_OPE_LE_DUMMY ope)
	{
		this->setOpeStrFromId(UT_OPE_LE);
	}
	void calcResult(T_UT_OPE_UNKNOWN_DUMMY ope)
	{
		this->setOpeStrFromType(ope);
		obj.setResult(false);
		obj.setHasResult(false);
	}
	void calcResult(T_UT_OPE_EQ_DUMMY ope)
	{
		this->setOpeStrFromType(ope);
		this->setResult(this->m_value == this->m_expect);
		this->setHasResult(true);
	}
	void calcResult(T_UT_OPE_NE_DUMMY ope)
	{
		this->setOpeStrFromType(ope);
		this->setResult(this->m_value != this->m_expect);
		this->setHasResult(true);
	}
	void calcResult(T_UT_OPE_GT_DUMMY ope)
	{
		this->setOpeStrFromType(ope);
		this->setResult(this->m_value > this->m_expect);
		this->setHasResult(true);
	}
	void calcResult(T_UT_OPE_GE_DUMMY ope)
	{
		this->setOpeStrFromType(ope);
		this->setResult(this->m_value >= this->m_expect);
		this->setHasResult(true);
	}
	void calcResult(T_UT_OPE_LT_DUMMY ope)
	{
		this->setOpeStrFromType(ope);
		this->setResult(this->m_value < this->m_expect);
		this->setHasResult(true);
	}
```

```
                void calcResult(T_UT_OPE_LE_DUMMY ope)
                {
                        this->setOpeStrFromType(ope);
                        this->setResult(this->m_value <= this->m_expect);
                        this->setHasResult(true);
                }

        private:
                T m_value;
                T m_expect;
                bool m_hasValue;
                bool m_hasExpect;
        };
        template<typename T, typename OPE>
        static CExprCResultObj<T>* makeResultObj2(const T value, typename T expect, OPE ope, const char* expr_str,
                                                                                                const char* expect_str)
        {
                CExprCResultObj<T>* obj = new CExprCResultObj<T>(value, expect);
                obj->setValue(value);
                obj->setExpect(expect);
                obj->setExprStr(expr_str);
                obj->setExpectStr(expect_str);
                obj->calcResult(ope);
                return obj;
        }
        template<typename OPE>
        static CExprCResultObj<int>* makeResultObj2ex(OPE ope, const char* expr_str, const char* expect_str)
        {
                CExprCResultObj<int>* obj = new CExprCResultObj<int>();
                obj->setExprStr(expr_str);
                obj->setExpectStr(expect_str);
                obj->setOpeStrFromType(ope);
                obj->setResult(false);
                obj->setHasResult(false);
                return obj;
        }
        template<typename T>
        static CExprCResultObj<T>* makeResultObj1(const T value, const char* expr_str)
        {
                CExprCResultObj<T>* obj = new CExprCResultObj<T>(value);
                obj->setExprStr(expr_str);
                obj->setResult(false);
                obj->setHasResult(false);
                return obj;
        }
        static CExprCResultObj<int>* makeResultObj1ex(const char* expr_str)
        {
                CExprCResultObj<int>* obj = new CExprCResultObj<int>();
                obj->setExprStr(expr_str);
                obj->setResult(false);
                obj->setHasResult(false);
                return obj;
        }
        static void outputUTResult(const bool is_child, int* passed, int* missed, CElapsedTime* elapsed_time,
                                                                                        CExprCResultObjBase* result_obj);
};

//処理時間計測
class CElapsedTime
{
public:
        typedef double TIMERCOUNT;
public:
        CElapsedTime();
        ~CElapsedTime();
```

```
	public:
		TIMERCOUNT getBeginTime() const { return this->m_beginTime; }
		TIMERCOUNT getEndTime() const { return this->m_endTime; }
		TIMERCOUNT getElapsedTime() const { return this->m_elapsedTime; }
		double getResult() const { return static_cast<double>(this->m_elapsedTime); }//経過時間（秒）
		double getResultMS() const { return static_cast<double>(this->m_elapsedTime) * 1000.0; }//経過時間（ミリ秒）
	public:
		TIMERCOUNT finish();
	private:
		TIMERCOUNT getTimer();
	private:
		TIMERCOUNT m_beginTime;
		TIMERCOUNT m_endTime;
		TIMERCOUNT m_elapsedTime;
	};
}; //namespace UnitTest

#define UT_BEGIN(module_name, group_id, attr) ¥
namespace __UnitTest_module_##module_name##__ ¥
{ ¥
	class __CUnitTestModle ¥
	{ ¥
	public: ¥
	__CUnitTestModle() ¥
		{ ¥
			UnitTest::CCollection::addFuncInfo(runUnitTest, #module_name, group_id, attr); ¥
		} ¥
	~__CUnitTestModle() ¥
		{ ¥
		} ¥
	public: ¥
		static int runUnitTest(int& __passed, int& __missed) ¥
		{ ¥
			const char* __module_name = #module_name; ¥
			const int __group_id = group_id; ¥
			UnitTest::CElapsedTime __elapsed_time; ¥
			const UnitTest::T_UT_ATTR __attr = attr; ¥
			__passed = 0; ¥
			__missed = 0; ¥
			UnitTest::CCollection::outputRunUTModuleBegin(__module_name, __group_id, __attr);
#define UT_END() ¥
			__elapsed_time.finish(); ¥
			UnitTest::CCollection::outputRunUTModuleEnd(__module_name, __group_id, __attr, __passed, __missed, ¥
                                                                __elapsed_time.getResultMS()); ¥
			return __missed; ¥
		} ¥
	}; ¥
	static __CUnitTestModle __unit_test_obj; ¥
}; //namespace __UnitTest_##module_name
#define _UT_PRINT_EXPR_AND_RESULT(is_child, expr, ope, expect) ¥
{ ¥
	UnitTest::CCollection::CExprCResultObjBase* ___result_obj = nullptr; ¥
	try ¥
	{ ¥
		UnitTest::CElapsedTime ___elapsed_time; ¥
		___result_obj = UnitTest::CCollection::makeResultObj2(expr, expect, ope, #expr, #expect); ¥
		___elapsed_time.finish(); ¥
		UnitTest::CCollection::outputUTResult(is_child, &__passed, &__missed, &___elapsed_time, ___result_obj); ¥
	 } ¥
	catch(std::exception const &e) ¥
	{ ¥
		if(!___result_obj) ¥
		{ ¥
			___result_obj = UnitTest::CCollection::makeResultObj2ex(ope, #expr, #expect); ¥
		} ¥
```

```
            ___result_obj->setException(e); ¥
            UnitTest::CCollection::outputUTResult(is_child, nullptr, &__missed, nullptr, ___result_obj); ¥
      } ¥
      if(___result_obj) ¥
      { ¥
            delete ___result_obj; ¥
      } ¥
}
#define _UT_PRINT_EXPR_AND_VALUE(is_child, expr) ¥
{ ¥
      UnitTest::CCollection::CExprCResultObjBase* ___result_obj = nullptr; ¥
      Try ¥
      { ¥
            UnitTest::CElapsedTime ___elapsed_time; ¥
            ___result_obj = UnitTest::CCollection::makeResultObj1(expr, #expr); ¥
            ___elapsed_time.finish(); ¥
            ___result_obj->setResult(true); ¥
            UnitTest::CCollection::outputUTResult(is_child, nullptr,   nullptr,   &___elapsed_time, ___result_obj); ¥
      } ¥
      catch(std::exception const &e) ¥
      { ¥
            if(!___result_obj) ¥
            { ¥
                  ___result_obj = UnitTest::CCollection::makeResultObj1ex(#expr); ¥
            } ¥
            ___result_obj->setException(e); ¥
            UnitTest::CCollection::outputUTResult(is_child, nullptr, &__missed, nullptr, ___result_obj); ¥
      } ¥
      if(___result_obj) ¥
      { ¥
            delete ___result_obj; ¥
      } ¥
}
#define _UT_PRINT_EXPR(is_child, expr) ¥
{ ¥
      UnitTest::CCollection::CExprCResultObjBase* ___result_obj = nullptr; ¥
      Try ¥
      { ¥
            ___result_obj = UnitTest::CCollection::makeResultObj1ex(#expr); ¥
            ___result_obj->setResult(true); ¥
            UnitTest::CCollection::outputUTResult(is_child, nullptr,   nullptr,   nullptr,          ___result_obj); ¥
      } ¥
      catch(std::exception const &e) ¥
      { ¥
            if(!___result_obj) ¥
            { ¥
                  ___result_obj = UnitTest::CCollection::makeResultObj1ex(#expr); ¥
            } ¥
            ___result_obj->setException(e); ¥
            UnitTest::CCollection::outputUTResult(is_child, nullptr, &__missed, nullptr, ___result_obj); ¥
      } ¥
      if(___result_obj) ¥
      { ¥
            delete ___result_obj; ¥
      } ¥
}

#define UT_EXPECT(expr, ope, expect)       _UT_PRINT_EXPR_AND_RESULT(false, expr, ope, expect);
#define UT_EXPECT_CHILD(expr, ope, expect) _UT_PRINT_EXPR_AND_RESULT(true,  expr, ope, expect);
#define UT_EXPR_WITH_RET(expr)             _UT_PRINT_EXPR_AND_VALUE(false, expr)
#define UT_EXPR_WITH_RET_CHILD(expr)       _UT_PRINT_EXPR_AND_VALUE(true,  expr)
#define UT_EXPR(expr)                      _UT_PRINT_EXPR(false, expr)
#define UT_EXPR_CHILD(expr)                _UT_PRINT_EXPR(true, expr)

#define UT_EXPECT_EQ(expr, expect)       UT_EXPECT(expr, UnitTest::T_UT_OPE_EQ_DUMMY(), expect)
```

```
#define UT_EXPECT_NE(expr, expect)       UT_EXPECT(expr, UnitTest::T_UT_OPE_NE_DUMMY(), expect)
#define UT_EXPECT_GT(expr, expect)       UT_EXPECT(expr, UnitTest::T_UT_OPE_GT_DUMMY(), expect)
#define UT_EXPECT_GE(expr, expect)       UT_EXPECT(expr, UnitTest::T_UT_OPE_GE_DUMMY(), expect)
#define UT_EXPECT_LT(expr, expect)       UT_EXPECT(expr, UnitTest::T_UT_OPE_LT_DUMMY(), expect)
#define UT_EXPECT_LE(expr, expect)       UT_EXPECT(expr, UnitTest::T_UT_OPE_LE_DUMMY(), expect)
#define UT_EXPECT_EQ_CHILD(expr, expect) UT_EXPECT_CHILD(expr, UnitTest::T_UT_OPE_EQ_DUMMY(), expect);
#define UT_EXPECT_NE_CHILD(expr, expect) UT_EXPECT_CHILD(expr, UnitTest::T_UT_OPE_NE_DUMMY(), expect);
#define UT_EXPECT_GT_CHILD(expr, expect) UT_EXPECT_CHILD(expr, UnitTest::T_UT_OPE_GT_DUMMY(), expect);
#define UT_EXPECT_GE_CHILD(expr, expect) UT_EXPECT_CHILD(expr, UnitTest::T_UT_OPE_GE_DUMMY(), expect);
#define UT_EXPECT_LT_CHILD(expr, expect) UT_EXPECT_CHILD(expr, UnitTest::T_UT_OPE_LT_DUMMY(), expect);
#define UT_EXPECT_LE_CHILD(expr, expect) UT_EXPECT_CHILD(expr, UnitTest::T_UT_OPE_LE_DUMMY(), expect);

#ifndef UT_TARGET_MODULE
#define UT_TARGET_MODULE nullptr
#endif//UT_TARGET_MODULE
#ifndef UT_TARGET_GROUP
#define UT_TARGET_GROUP 0
#endif//UT_TARGET_GROUP
#define UT_RUN_ALL(attr) UnitTest::CCollection::runUnitTest(nullptr, 0, attr)
#define UT_RUN_MODULE(module_name, attr) UnitTest::CCollection::runUnitTest(module_name, 0, attr)
#define UT_RUN_GROUP(group_id, attr) UnitTest::CCollection::runUnitTest(nullptr, group_id, attr)
#define UT_RUN_STANDARD(attr) UnitTest::CCollection::runUnitTestStandard(attr)
#define UT_RETURN() {return UnitTest::CCollection::getLastMissedTotal();}
#define UT_RETURN_WHEN_MISSED() ¥
{ ¥
     if(UnitTest::CCollection::getLastMissedTotal() > 0) ¥
     { ¥
          return UnitTest::CCollection::getLastMissedTotal(); ¥
     } ¥
}
#define UT_EXIT_WHEN_MISSED() ¥
{ ¥
     if(UnitTest::CCollection::getLastMissedTotal() > 0) ¥
     { ¥
          exit(UnitTest::CCollection::getLastMissedTotal()); ¥
     } ¥
}
#define UT_ABORT_WHEN_MISSED() ¥
{ ¥
     if(UnitTest::CCollection::getLastMissedTotal() > 0) ¥
     { ¥
          abort(); ¥
     } ¥
}
#define UT_ASSERT_WHEN_MISSED() ASSERT(UnitTest::CCollection::getLastMissedTotal() == 0)
#ifdef UT_AUTO
#define UT_RUN_MAIN(result_var) ¥
     UnitTest::CCollection::runUnitTestStandard(UnitTest::UT_ATTR_AUTO); ¥
     result_var = UnitTest::CCollection::getLastMissedTotal(); ¥
     UT_RETURN()
#else//UT_AUTO
#define UT_RUN_MAIN(result_var)
#endif//UT_AUTO
#define UT_RESET_LAST_RESULT() UnitTest::CCollection::resetLastResult()
#define UT_LAST_PASSED_TOTAL() UnitTest::CCollection::getLastPassedTotal()
#define UT_LAST_MISSED_TOTAL() UnitTest::CCollection::getLastMissedTotal()
#define UT_SET_OUTPUT_FUNC(func) UnitTest::CCollection::setOutputFunc(func)
#define UT_OUTPUT(fmt, ...) UnitTest::CCollection::output(fmt, __VA_ARGS__)
#define UT_WITH_(statement) statement

#else//UT_ENABLED

#define UT_BEGIN(module_name, group_id, attr) ¥
namespace __UnitTest_module_##module_name##__ ¥
{ ¥
```

```
	template<class T> ¥
	class __CUnitTestModle_dummy ¥
	{ ¥
	private: ¥
		void __dummy(T dummy) ¥
		{ ¥
			if (false) ¥
			{
#define UT_END() ¥
			} ¥
		} ¥
	}; ¥
}; //namespace __UnitTest_##module_name
#define UT_PRINT_RESULT(is_child, expr, value)
#define UT_PRINT_EXPR(is_child, expr)
#define UT_PRINT_EXPR_WITH_RET(is_child, expr)
#define UT_CLAC_AND_PRINT_RESULT(is_child, expr, value, ope, expect)
#define UT_EXPECT(expr, ope, expect)
#define UT_EXPECT_CHILD(expr, ope, expect)
#define UT_EXPR(expr)
#define UT_EXPR_CHILD(expr)
#define UT_EXPR_WITH_RET(expr)
#define UT_EXPR_WITH_RET_CHILD(expr)
#define UT_EXPECT_EQ(expr, expect)
#define UT_EXPECT_NE(expr, expect)
#define UT_EXPECT_GT(expr, expect)
#define UT_EXPECT_GE(expr, expect)
#define UT_EXPECT_LT(expr, expect)
#define UT_EXPECT_LE(expr, expect)
#define UT_EXPECT_EQ_CHILD(expr, expect)
#define UT_EXPECT_NE_CHILD(expr, expect)
#define UT_EXPECT_GT_CHILD(expr, expect)
#define UT_EXPECT_GE_CHILD(expr, expect)
#define UT_EXPECT_LT_CHILD(expr, expect)
#define UT_EXPECT_LE_CHILD(expr, expect)
#define UT_RUN_ALL(attr)
#define UT_RUN_MODULE(module_name, attr)
#define UT_RUN_GROUP(group_id, attr)
#define UT_RUN_STANDARD(attr)
#define UT_RETURN()
#define UT_RETURN_WHEN_MISSED()
#define UT_EXIT_WHEN_MISSED()
#define UT_ABORT_WHEN_MISSED()
#define UT_ASSERT_WHEN_MISSED()
#define UT_RUN_MAIN(result_var)
#define UT_RESET_LAST_RESULT()
#define UT_LAST_PASSED_TOTAL() 0
#define UT_LAST_MISSED_TOTAL() 0
#define UT_SET_OUTPUT_FUNC(func)
#define UT_OUTPUT(fmt, ...)
#define UT_WITH_(statement)

#endif//UT_ENABLED

#endif//__UNIT_TEST_H__
```

【unit_test.cpp】

```
#include "unit_test.h"

#ifdef UT_ENABLED

#include <stdio.h>

//for Windows
#include <windows.h> //コンソールのカラー表示用にインクルード
```

```
#include <conio.h>   //（同上）
#include <winbase.h> //パフォーマンスカウンターの計測用にインクルード

namespace UnitTest
{
	//ユニットテスト情報
	static const int UNIT_TEST_FUNC_LIST_NUM_MAX = 1024;
	static int s_funcListNum = 0;
	static CCollection::UNIT_TEST_FUNC_INFO s_funcList[UNIT_TEST_FUNC_LIST_NUM_MAX];
	static int s_lastPassedTotal = 0;
	static int s_lastMissedTotal = 0;
	static int outputFuncDefault(const char* fmt, va_list list)
	{
		return vfprintf(stdout, fmt, list);
	}
	static CCollection::UNIT_TEST_OUTPUT_FUNC_P s_outputFunc = outputFuncDefault;

	//ユニットテストメインクラス

	//ユニットテスト登録
	bool CCollection::addFuncInfo(UNIT_TEST_FUNC_P func, const char* module_name, const int group_id,
	                                                                                const T_UT_ATTR attr)
	{
		if (s_funcListNum >= UNIT_TEST_FUNC_LIST_NUM_MAX)
		{
			return false;
		}
		UNIT_TEST_FUNC_INFO* info = &s_funcList[s_funcListNum++];
		info->m_func = func;
		info->m_moduleName = module_name;
		info->m_groupId = group_id;
		info->m_attr = attr;
		info->passed = 0;
		info->missed = 0;
		info->elapsed_time = 0.;
		return true;
	}

	//アクセッサ
	int CCollection::getFuncInfoListNum()
	{
		return s_funcListNum;
	}
	const CCollection::UNIT_TEST_FUNC_INFO* CCollection::getFuncInfoListTop()
	{
		return s_funcList;
	}
	const CCollection::UNIT_TEST_FUNC_INFO* CCollection::getFuncInfo(const int index)
	{
		return index >= 0 && index < s_funcListNum ? &s_funcList[index] : nullptr;
	}
	const CCollection::UNIT_TEST_FUNC_INFO* CCollection::getFuncInfo(const char* module_name)
	{
		UNIT_TEST_FUNC_INFO* info = s_funcList;
		for (int i = 0; i < s_funcListNum; ++i, ++info)
		{
			if (strcmp(module_name, info->m_moduleName) == 0)
			{
				return  &s_funcList[i];
			}
		}
		return nullptr;
	}
	int CCollection::getLastPassedTotal()
	{
```

```
        return s_lastPassedTotal;
    }
    int CCollection::getLastMissedTotal()
    {
        return s_lastMissedTotal;
    }

    //前回の実行結果をリセット
    void CCollection::resetLastResult()
    {
        s_lastPassedTotal = 0;
        s_lastMissedTotal = 0;
        UNIT_TEST_FUNC_INFO* info = s_funcList;
        for (int i = 0; i < s_funcListNum; ++i, ++info)
        {
            info->passed = 0;
            info->missed = 0;
            info->elapsed_time = 0.;
        }
    }

    //ユニットテスト実行
    int CCollection::runUnitTest(const char* target_module_name, const int target_group_id,
                                                                    const T_UT_ATTR target_attr)
    {
        int passed_total = 0;
        int missed_total = 0;
        CElapsedTime elapsed_time_total;
        outputRunUTBegin(target_module_name, target_group_id, target_attr);
        UNIT_TEST_FUNC_INFO* info = s_funcList;
        for (int i = 0; i < s_funcListNum; ++i, ++info)
        {
            if ((target_module_name == nullptr ||
                (target_module_name != nullptr && info->m_moduleName != nullptr &&
                    strcmp(target_module_name, info->m_moduleName) == 0)) &&
                (target_group_id == 0 || (target_group_id != 0 && target_group_id == info->m_groupId)) &&
                (target_attr == UT_ATTR_ANY || (target_attr != UT_ATTR_ANY &&
                                                (target_attr & info->m_attr) != UT_ATTR_NONE)))
            {
                int passed = 0;
                int missed = 0;
                CElapsedTime elapsed_time;
                info->m_func(passed, missed);
                elapsed_time.finish();
                passed_total += passed;
                missed_total += missed;
                info->passed = passed;
                info->missed = missed;
                info->elapsed_time = elapsed_time.getResultMS();
            }
        }
        elapsed_time_total.finish();
        s_lastPassedTotal = passed_total;
        s_lastMissedTotal = missed_total;
        outputRunUTEnd(target_module_name, target_group_id, target_attr, passed_total, missed_total,
                                                                    elapsed_time_total.getResultMS());
        return missed_total;
    }
    int CCollection::runUnitTestStandard(const T_UT_ATTR target_attr)
    {
        return runUnitTest(UT_TARGET_MODULE, UT_TARGET_GROUP, target_attr);
    }

    //ユニットテスト結果表示
#if 0
```

```
	//for Unix
	#define _COLOR_BEGIN()
	#define _COLOR_RESET() output("¥x1b[0m")
	#define _COLOR_NORMAL() output("¥x1b[40m¥x1b[37m")
	#define _COLOR_OK() output("¥x1b[40m¥x1b[34m")
	#define _COLOR_NG() output("¥x1b[41m¥x1b[37m")
	#define _COLOR_UNKNOWN() output("¥x1b[44m¥x1b[31m")
	#define _COLOR_EXPR() output("¥x1b[40m¥x1b[32m")
	#define _COLOR_OPE() output("¥x1b[40m¥x1b[37m")
	#define _COLOR_EXPECT() output("¥x1b[40m¥x1b[32m")
	#define _COLOR_VALUE() output("¥x1b[40m¥x1b[32m")
	#define _COLOR_ELAPSED_TIME() output("¥x1b[40m¥x1b[32m")
	#define _COLOR_EXCEPTION() output("¥x1b[41m¥x1b[37m")
	#define _COLOR_END()
#else
	//for Windows
	CONSOLE_SCREEN_BUFFER_INFO _csb_info_backuped;
	HANDLE       _hStdout = INVALID_HANDLE_VALUE;
	#define _COLOR_BEGIN() ¥
	{ ¥
		_hStdout = GetStdHandle(STD_OUTPUT_HANDLE); ¥
		GetConsoleScreenBufferInfo(_hStdout, &_csb_info_backuped); ¥
	}
	#define _COLOR_RESET() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, _csb_info_backuped.wAttributes); ¥
	}
	#define _COLOR_NORMAL() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_BLUE|FOREGROUND_GREEN|FOREGROUND_RED|FOREGROUND_INTENSITY); ¥
	}
	#define _COLOR_OK() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_BLUE|FOREGROUND_GREEN|FOREGROUND_RED|FOREGROUND_INTENSITY| ¥
		                                                                BACKGROUND_BLUE|BACKGROUND_INTENSITY); ¥
	}
	#define _COLOR_NG() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_BLUE|FOREGROUND_GREEN|FOREGROUND_RED|FOREGROUND_INTENSITY| ¥
		                                                                BACKGROUND_RED|BACKGROUND_INTENSITY); ¥
	}
	#define _COLOR_UNKOWN() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_RED|FOREGROUND_INTENSITY|BACKGROUND_GREEN| ¥
		BACKGROUND_RED|BACKGROUND_INTENSITY); ¥
	}
	#define _COLOR_EXPR() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_GREEN|FOREGROUND_INTENSITY); ¥
	}
	#define _COLOR_OPE() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_GREEN|FOREGROUND_RED|FOREGROUND_INTENSITY); ¥
	}
	#define _COLOR_EXPECT() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_GREEN|FOREGROUND_INTENSITY); ¥
	}
	#define _COLOR_VALUE() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_GREEN|FOREGROUND_INTENSITY); ¥
	}
	#define _COLOR_ELAPSED_TIME() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, FOREGROUND_RED|FOREGROUND_GREEN|FOREGROUND_INTENSITY); ¥
```

```
	}
	#define _COLOR_EXCEPTION() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, BACKGROUND_BLUE|BACKGROUND_RED|BACKGROUND_INTENSITY); ¥
	}
	#define _COLOR_END() ¥
	{ ¥
		SetConsoleTextAttribute(_hStdout, _csb_info_backuped.wAttributes); ¥
		_hStdout = INVALID_HANDLE_VALUE; ¥
	}
#endif
	void CCollection::setOutputFunc(UNIT_TEST_OUTPUT_FUNC_P func)
	{
		s_outputFunc = func;
	}
	int CCollection::output(const char* fmt, ...)
	{
		if (!s_outputFunc)
			return 0;
		va_list list;
		va_start(list, fmt);
		const int ret = s_outputFunc(fmt, list);
		va_end(list);
		return ret;
	}
	void CCollection::outputRunUTBegin(const char* target_module_name, const int target_group_id,
	                                   const UnitTest::T_UT_ATTR target_attr)
	{
		_COLOR_BEGIN();
		_COLOR_RESET();
		_COLOR_NORMAL();
		output("¥n");
		output("Start unit test: ");
		if (target_module_name)
		{
			output("[Target module=¥"%s¥"]", target_module_name);
		}
		if (target_group_id != 0)
		{
			output("[Target group=%d]", target_group_id);
		}
		if (!target_module_name && target_group_id == 0)
		{
			output("[Target=All]");
		}
		if (target_attr != UT_ATTR_ANY)
		{
			output("[Attr=0x%08x]", target_attr);
		}
		else
		{
			output("[Attr=ANY]");
		}
		output("¥n");
		output("============================================================¥n");
		_COLOR_END();
	}
	void CCollection::outputRunUTEnd(const char* target_module_name, const int target_group_id,
	                                 const UnitTest::T_UT_ATTR target_attr, const int passed_total,
	                                 const int missed_total, const double elapsed_time_total)
	{
		_COLOR_BEGIN();
		_COLOR_RESET();
		_COLOR_NORMAL();
		output("¥n");
```

```
		output("============================================================¥n");
		output("Finish unit test: Total [test=%d, passed=", passed_total + missed_total);
		if (passed_total > 0)
		{
			_COLOR_OK();
			output("%d", passed_total);
		}
		else
		{
			_COLOR_NORMAL();
			output("%d", passed_total);
		}
		_COLOR_NORMAL();
		output(", missed=");
		if (missed_total > 0)
		{
			_COLOR_NG();
			output("%d", missed_total);
		}
		else
		{
			_COLOR_NORMAL();
			output("%d", missed_total);
		}
		_COLOR_NORMAL();
		output("] ");
		_COLOR_ELAPSED_TIME();
		output("%.6lf ms", elapsed_time_total);
		_COLOR_NORMAL();
		output(" -----¥n");
		output("¥n");
		_COLOR_END();
	}
	void CCollection::outputRunUTModuleBegin(const char* module_name, const int group_id,
	                                         const UnitTest::T_UT_ATTR attr)
	{
		_COLOR_BEGIN();
		_COLOR_RESET();
		_COLOR_NORMAL();
		output("¥n");
		output("----- Start unit test module: ¥"%s¥" (Group=%d,Attr=0x%08x) -----¥n", module_name, group_id, attr);
		_COLOR_END();
	}
	void CCollection::outputRunUTModuleEnd(const char* module_name, const int group_id, const UnitTest::T_UT_ATTR attr,
	                                       const int passed, const int missed, const double elapsed_time)
	{
		_COLOR_BEGIN();
		_COLOR_RESET();
		_COLOR_NORMAL();
		output("----- Finish unit test module: [test=%d, passed=", passed + missed);
		if (passed > 0)
		{
			_COLOR_OK();
			output("%d", passed);
		}
		else
		{
			_COLOR_NORMAL();
			output("%d", passed);
		}
		_COLOR_NORMAL();
		output(", missed=");
		if (missed > 0)
		{
			_COLOR_NG();
```

```
                output("%d", missed);
        }
        else
        {
                _COLOR_NORMAL();
                output("%d", missed);
        }
        _COLOR_NORMAL();
        output("]");
        _COLOR_NORMAL();
        output(" ");
        _COLOR_ELAPSED_TIME();
        output("%.6lf ms", elapsed_time);
        _COLOR_NORMAL();
        output(" -----¥n");
        _COLOR_END();
}
void CCollection::outputUTResult(const bool is_child, int* passed, int* missed, CElapsedTime* elapsed_time,
                                                                                CExprCResultObjBase* result_obj)
{
        _COLOR_BEGIN();
        _COLOR_RESET();
        _COLOR_NORMAL();
        bool is_count_passed = false;
        bool is_count_missed = false;
        if (result_obj->hasOpeStr() && result_obj->hasExpectStr())
        {
                if (result_obj->hasResult())
                {
                        if (result_obj->getResult())
                        {
                                is_count_passed = true;
                                _COLOR_NORMAL();
                                output(" ");
                                _COLOR_OK();
                                output("[OK]");
                                _COLOR_NORMAL();
                                output(" ");
                        }
                        else
                        {
                                is_count_missed = true;
                                _COLOR_NG();
                                output("*[NG!]");
                        }
                }
                else
                {
                        is_count_missed = true;
                        _COLOR_UNKOWN();
                        output("*[??]");
                        _COLOR_NORMAL();
                        output(" ");
                }
                _COLOR_NORMAL();
                output(" <-- ");
        }
        else
        {
                _COLOR_NORMAL();
                output("      ");
                output("     ");
        }
        if (is_child)
        {
```

```
                _COLOR_NORMAL();
                output("    ");
        }
        if (result_obj->hasExprStr())
        {
                _COLOR_EXPR();
                output("%s", result_obj->getExprStr());
        }
        if (result_obj->hasOpeStr())
        {
                _COLOR_NORMAL();
                output(" ");
                _COLOR_OPE();
                output("%s", result_obj->getOpeStr());
                _COLOR_NORMAL();
                output(" ");
        }
        if (result_obj->hasExpectStr())
        {
                _COLOR_EXPECT();
                output("%s", result_obj->getExpectStr());
        }
        if (result_obj->hasValueStr())
        {
                _COLOR_NORMAL();
                output(" (ret=");
                _COLOR_VALUE();
                output("%s", result_obj->getValueStr());
                _COLOR_NORMAL();
                output(")");
        }
        if (elapsed_time)
        {
                _COLOR_NORMAL();
                output(" ");
                _COLOR_ELAPSED_TIME();
                output("%.6lf ms", elapsed_time->getResultMS());
        }
        if (result_obj->hasException())
        {
                is_count_missed = true;
                _COLOR_NORMAL();
                output(" ");
                _COLOR_EXCEPTION();
                output("<EXCEPTION!!");
                const char* msg = result_obj->getExceptionStr();
                if (msg)
                {
                     output(":%s", msg);
                }
                output(">");
        }
        _COLOR_RESET();
        output("¥n");
        _COLOR_END();

        if (is_count_passed && passed)
        {
                ++(*passed);
        }
        if (is_count_missed && missed)
        {
                ++(*missed);
        }
}
```

```
//処理時間計測
CElapsedTime::CElapsedTime() :
	m_beginTime(0.f),
	m_endTime(0.f),
	m_elapsedTime(0.f)
{
	this->m_beginTime = getTimer();
}
CElapsedTime::~CElapsedTime()
{
}
CElapsedTime::TIMERCOUNT CElapsedTime::finish()
{
	this->m_endTime = getTimer();
	this->m_elapsedTime = this->m_endTime - this->m_beginTime;
	return this->m_endTime;
}
CElapsedTime::TIMERCOUNT CElapsedTime::getTimer()
{
	//for Windows
	static bool freq_initialized = false;
	static LARGE_INTEGER freq;
	if (!freq_initialized)
	{
		QueryPerformanceFrequency(&freq);
		freq_initialized = true;
	}
	LARGE_INTEGER counter;
	QueryPerformanceCounter(&counter);
	const TIMERCOUNT timer = static_cast<TIMERCOUNT>(counter.QuadPart) / static_cast<TIMERCOUNT>(freq.QuadPart);
	return timer;
}

//ユニットテスト結果格納クラス
CCollection::CExprCResultObjBase::CExprCResultObjBase() :
	m_result(true),
	m_hasResult(false),
	m_hasException(false),
	m_hasExprStr(false),
	m_hasValueStr(false),
	m_hasOpeStr(false),
	m_hasExpectStr(false),
	m_exprStr(),
	m_valuieStr(),
	m_opeStr(),
	m_expectStr(),
	m_exceptionStr()
{
}
CCollection::CExprCResultObjBase::~CExprCResultObjBase()
{
}
void CCollection::CExprCResultObjBase::setException(std::exception const& e)
{
	this->m_hasException = true; this->m_exceptionStr = e.what();
}
void CCollection::CExprCResultObjBase::setExprStr(const char* expr)
{
	if (expr)
	{
		this->m_exprStr = expr;
		this->m_hasExprStr = true;
	}
}
```

```
	void CCollection::CExprCResultObjBase::setValueStr(const char* value)
	{
		if (value)
		{
			this->m_valuieStr = value;
			this->m_hasValueStr = true;
		}
	}
	void CCollection::CExprCResultObjBase::setOpeStr(const char* ope)
	{
		if (ope)
		{
			this->m_opeStr = ope;
			this->m_hasOpeStr = true;
		}
	}
	const char* CCollection::CExprCResultObjBase::setOpeStrFromId(const E_UT_OPE ope)
	{
		static const char* ope_str[UT_OPE_NUM] = { "??", "==", "!=", ">", ">=", "<", "<=" };
		this->setOpeStr(ope_str[ope]);
		return this->getOpeStr();
	}
	void CCollection::CExprCResultObjBase::setExpectStr(const char* expect)
	{
		if (expect)
		{
			this->m_expectStr = expect;
			this->m_hasExpectStr = true;
		}
	}
};

#endif//UT_ENABLED
```

▼ ユニットテスト処理サンプル

ユニットテストを行う処理のサンプルを示す。赤字はユニットテスト処理。

【unit_test_id.h】

```
#pragma once
#ifndef __UNIT_TEST_ID_H__
#define __UNIT_TEST_ID_H__

#include "unit_test.h"

#ifdef UT_ENABLED
//ユニットテストグループ ID
enum E_UT_GROUP
{
	UT_GROUP_NONE = 0,
	UT_GROUP_USER_A = 1,
	UT_GROUP_USER_B = 2,
};
#endif//UT_ENABLED

#endif//__UNIT_TEST_ID_H__
```

【moculde_a.cpp】

```
#include <exception>

class CModuleA
{
public:
```

```
	int func1(int p1, int p2)
	{
		return p1 + p2;
	}
	bool func2()
	{
		return true;
	}
	bool func3()
	{
		throw std::exception("Thrown message");
		return true;
	}
};

//ユニットテスト
#include "unit_test.h"
#include "game/unit_test_id.h"
UT_BEGIN(CModuleA, UT_GROUP_USER_A, UnitTest::UT_ATTR_AUTO)
//モジュール名とグループ ID、属性を指定 ※特定のモジュール名やグループ ID に絞ったテストを実行可能。
//                                      ※main から自動実行するには UT_ATTR_AUTO 属性を付与する必要がある。
//                                        属性は第 0 ビット目が自動実行用に予約されている以外は自由に使って良い。
//                                        テスト実行時に対象属性の指定があり、ビットマスク（&演算の結果が
//                                        0 以外かどうか）で実行の可否が判定される仕組み。
{
	CModuleA _CModuleA;
	UT_EXPECT_EQ(_CModuleA.func1(1, 2), 3);
	UT_EXPECT_EQ(_CModuleA.func1(3, 4), 7);
	UT_EXPECT_NE(_CModuleA.func1(4, 5), 9);
	UT_EXPECT_EQ(_CModuleA.func2(), true);
	UT_EXPECT_NE(_CModuleA.func2(), true);
	UT_EXPECT_EQ(_CModuleA.func3(), true);
}
UT_END()
```

【moculde_b.cpp】

```
class CModuleB
{
public:
	int func1(int p1, int p2, int p3)
	{
		return p1 * p2 * p3;
	}
	void func2(int p1, int p2, int& ret)
	{
		ret = p1 * p2;
	}
};

class CModuleC
{
public:
	float func(float p1, float p2, float& ret1, float& ret2)
	{
		ret1 = p1 + p2;
		ret2 = p1 - p2;
		return p1 * p2;
	}
};

//ユニットテスト
#include "unit_test.h"
#include "game/unit_test_id.h"
UT_BEGIN(CModuleB, UT_GROUP_USER_B, UnitTest::UT_ATTR_AUTO)
//モジュール名とグループ ID、属性を指定 ※特定のモジュール名やグループ ID に絞ったテストを実行可能。
```

```
//                                  ※main から自動実行するには UT_ATTR_AUTO 属性を付与する必要がある。
//                                    属性は第 0 ビット目が自動実行用に予約されている以外は自由に使って良い。
//                                    テスト実行時に対象属性の指定があり、ビットマスク（&演算の結果が 0 以外か
//                                    どうか）で実行の可否が判定される仕組み。
{
    CModuleB _CModuleB;
    int out_par = 0;
    UT_EXPECT_EQ(_CModuleB.func1(1, 2, 3), 1+2+3);
    UT_EXPECT_NE(_CModuleB.func1(3, 4, 5), 60);
    UT_EXPECT_GT(_CModuleB.func1(5, 6, 7), 10);
    UT_EXPECT_GE(_CModuleB.func1(7, 8, 9), 20);
    UT_EXPECT_LT(_CModuleB.func1(9, 10, 11), 30);
    UT_EXPECT_LE(_CModuleB.func1(11, 12, 13), 40);
    UT_EXPR(_CModuleB.func2(1, 2, out_par)); UT_EXPECT_EQ_CHILD(out_par, 2);
    UT_EXPR(_CModuleB.func2(3, 4, out_par)); UT_EXPECT_NE_CHILD(out_par, 12);
}
UT_END()
UT_BEGIN(CModuleC, UT_GROUP_USER_B, UnitTest::UT_ATTR_NOAUTO)//モジュール名とグループ ID、属性を指定
{
    CModuleC _CModuleC;
    float out_par1 = 0.f;
    float out_par2 = 0.f;
    UT_EXPR_WITH_RET(_CModuleC.func(1.1f, 2.2f, out_par1, out_par2));
                                        UT_EXPECT_EQ_CHILD(out_par1, 2.f); UT_EXPECT_LT_CHILD(out_par2, 0.f);
    UT_EXPR_WITH_RET(_CModuleC.func(3.3f, 4.4f, out_par1, out_par2));
                                        UT_EXPECT_NE_CHILD(out_par1, 7.f); UT_EXPECT_GT_CHILD(out_par2, 0.f);
    UT_EXPR_WITH_RET(_CModuleC.func(5.5f, 6.6f, out_par1, out_par2));
                                        UT_EXPECT_LE_CHILD(out_par1, 20.f); UT_EXPECT_GE_CHILD(out_par2, 10.f);
    UT_EXPR_WITH_RET(_CModuleC.func(7.7f, 8.8f, out_par1, out_par2));
                                        UT_EXPECT_LT_CHILD(out_par1, 1.f); UT_EXPECT_LE_CHILD(out_par2, 2.f);
}
UT_END()
```

【game_main.h】

```
#pragma once
#ifndef __GAME_MAIN_H__
#define __GAME_MAIN_H__

//ゲームメインクラス
class CGameMain
{
public:
    //コンストラクタ
    explicit CGameMain(const int argc, const char* argv[]);
private:
    explicit CGameMain() :m_argc(0), m_argv(nullptr){}//デフォルトコンストラクタ無効化（private 化）
    explicit CGameMain(CGameMain&) : m_argc(0), m_argv(nullptr){}//コピーコンストラクタ無効化（private 化）
public:
    //デストラクタ
    ~CGameMain();
public:
    //アクセッサ
    int getArgc() const { return this->m_argc; }
    const char* getArgv(const int index) const
    { return index >= 0 && index < this->m_argc ? this->m_argv[index] : nullptr; }
    int getInitializeResult() const { return this->m_initializeResult; }
    int getFinalizeResult() const { return this->m_finalizeResult; }
    int getMainResult() const { return this->m_mainResult; }
    operator int() const { return this->m_mainResult; }//キャスト演算子オーバーロード

public:
    //初期化処理
    int initialize();

    //終了処理
```

```
	int finalize();

	//メイン処理
	int main();

private:
	//実行時パラメータ
	const int m_argc;
	const char** m_argv;

	//処理結果
	int m_initializeResult;
	int m_finalizeResult;
	int m_mainResult;
};

#endif//__GAME_MAIN_H__
```

【game_main.cpp】

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

#include "game/game_main.h"
#include "unit_test.h"
#include "game/unit_test_id.h"

//ゲームメインクラス

//コンストラクタ
CGameMain::CGameMain(const int argc, const char* argv[]):
	m_argc(argc),
	m_argv(argv),
	m_initializeResult(EXIT_SUCCESS),
	m_finalizeResult(EXIT_SUCCESS),
	m_mainResult(EXIT_SUCCESS)
{
	printf("- CGameMain::CGameMain()¥n");
}

//デストラクタ
CGameMain::~CGameMain()
{
	printf("- CGameMain::~CGameMain()¥n");
}

//初期化処理
int CGameMain::initialize()
{
	printf("- CGameMain::initialize() [BEGIN] ...¥n");
	this->m_initializeResult = EXIT_SUCCESS;
	printf("- CGameMain::initialize() [END]¥n");
	return this->m_initializeResult;
}

//終了処理
int CGameMain::finalize()
{
	printf("- CGameMain::finalize() [BEGIN] ...¥n");
	this->m_finalizeResult = EXIT_SUCCESS;
	printf("- CGameMain::finalize() [END]¥n");
	return this->m_finalizeResult;
}

//メイン処理
int CGameMain::main()
```

```
{
    printf("- CGameMain::main() [BEGIN] ...¥n");

    //自動ユニットテスト
    //※UT_ENABLED と UT_AUTO が定義されている時だけ処理が有効化される
    //  UT_TARGET_MODULE と UT_TARGET_GROUP の定義の影響を受けて、実行されるユニットテストが限定される。
    //  メイン処理の最初に実行されるようにしておく。メモリ管理などの基本的な初期化処理が終わった後が良い。
    //※ユニットテストの結果に一つでもミスがあったら return する。return 値はミス数。
    //  その前に、パラメータで指定された変数に同結果値を格納する。
    //  この return 値を main 関数の return 値にする。
    //  これにより、jenkins などのツールからユニットテストの結果をハンドリングできる。
    UT_RUN_MAIN(this->m_mainResult);

#if 1
    //手動ユニットテスト
    //※UT_ENABLED が定義されている時だけ処理が有効化される
    //  プログラム中の任意の場所から呼び出せる。
    UT_OUTPUT("手動ユニットテスト:ALL¥n");
    UT_RUN_ALL(UnitTest::UT_ATTR_ANY);              //UT_AUTO と無関係に全モジュールのテストを実行するマクロ
                                                    //※実行対象属性を指定する。テスト登録されている関数の属性との
                                                    //  ビットマスク（両者の&演算の結果が 0 以外かどうか）によって
                                                    //  実行対象がマスクされる。
    UT_OUTPUT("手動ユニットテスト:MODULE(¥"CModuleB¥")¥n");
    UT_RUN_MODULE("CModuleB", UnitTest::UT_ATTR_ANY); //UT_AUTO と無関係に指定のモジュールのテストを実行するマクロ
    UT_OUTPUT("手動ユニットテスト:GROUP(2)¥n");
    UT_RUN_GROUP(2, UnitTest::UT_ATTR_ANY);         //UT_AUTO と無関係に指定のグループのテストを実行するマクロ
    UT_OUTPUT("手動ユニットテスト:STANDARD¥n");
    UT_RUN_STANDARD(UnitTest::UT_ATTR_AUTO);        //UT_AUTO と無関係にモジュールのテストを実行するマクロ
                                                    //※UT_TARGET_MODULE と UT_TARGET_GROUP の定義の影響を受けて、
                                                    //  実行されるユニットテストが限定される。
#endif

#if 1
    //前回のユニットテストの結果を確認／リセット／結果に基づいて終了
    UT_OUTPUT("手動ユニットテスト結果：passed=%d, missed=%d¥n", UT_LAST_PASSED_TOTAL(), UT_LAST_MISSED_TOTAL());
                                                                        //前回のユニットテストの結果を取得
    UT_RESET_LAST_RESULT();   //前回のユニットテストの結果をリセット
    UT_OUTPUT("結果をリセット：passed=%d, missed=%d¥n", UT_LAST_PASSED_TOTAL(), UT_LAST_MISSED_TOTAL());
    UT_RETURN_WHEN_MISSED();  //前回のユニットテストの結果に一つでもミスがあったら return する
    UT_EXIT_WHEN_MISSED();    //前回のユニットテストの結果に一つでもミスがあったら exit する
    UT_ABORT_WHEN_MISSED();   //前回のユニットテストの結果に一つでもミスがあったら abort する
    UT_ASSERT_WHEN_MISSED();  //前回のユニットテストの結果に一つでもミスがあったらアサーション違反とする
#endif

#if 1
    //ユニットテストのサポート処理
    UT_OUTPUT("UT_SET_OUTPUT_FUNC()：実行前¥n");
    UT_SET_OUTPUT_FUNC(nullptr); //ユニットテストの表示用関数を変更する（ターゲットプラットフォームに合わせた
                                 //関数にするなど）
                                 //※型:int(*UNIT_TEST_OUTPUT_FUNC_P)(const char* fmt, va_list list)
                                 //  nullptr を指定すると一切何も表示されなくなる。
    UT_OUTPUT("UT_SET_OUTPUT_FUNC()：実行後¥n");
    UT_WITH_(printf("UT_WITH_()マクロを使ってプリント¥n")); //※UT_WITH_()マクロ内に書いた処理は、
                                                          //  UT_ENABLED 有効時のみ実行される。
#endif

    //以下、本来のゲームメイン処理
    printf("本来のゲームメイン処理 ...¥n");

    this->m_mainResult = EXIT_SUCCESS;
    printf("- CGameMain::main() [END]¥n");
    return this->m_mainResult;
}
```

【main.cpp】

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

#include "game/game_main.h"

int main(const int argc, const char* argv[])
{
	printf("- main() [BEGIN] ...¥n");

	CGameMain* game_main = new CGameMain(argc, argv);

	//初期化処理
	game_main->initialize();

	//メイン処理
	game_main->main();

	//終了処理
	game_main->finalize();

	printf("- main() [END]¥n");

	//main 関数終了
//	exit(*game_main);  //exit() を実行するとデストラクタが実行されないので、
	return *game_main; //やむを得ない中断時以外はなるべく main() 関数の return を使う事。
	                   //なお、どちらを使用しても、環境変数 %ERRORLEVEL% には、きちんと結果が返される。
}
```

■■以上■■

■ 索引

索引項目が見つかりません。

ユニットテストと継続的ビルド

以　上