CppUTest で C/C++コードをテストする

TN0003-01／2024 年 04 月 05 日／こがねさん(著)
https://www.kumikomist.com/

■目次

■文書内の記号

	取り扱いにおける禁止事項（してはいけないこと）を示しています。
	取扱における指示事項（必ずしなければいけないこと）を示しています。

	取り扱いにおける注記事項を示しています。
	取り扱いにおけるポイントを示しています。

■改訂履歴

版数	日付	内容
00	2023 年 06 月 25 日	初版。
01	2024 年 04 月 05 日	『7. CppUMock』を追加。

1. はじめに

CppUTest は、単体テストを自動化するためのフリーソフトウェアです。

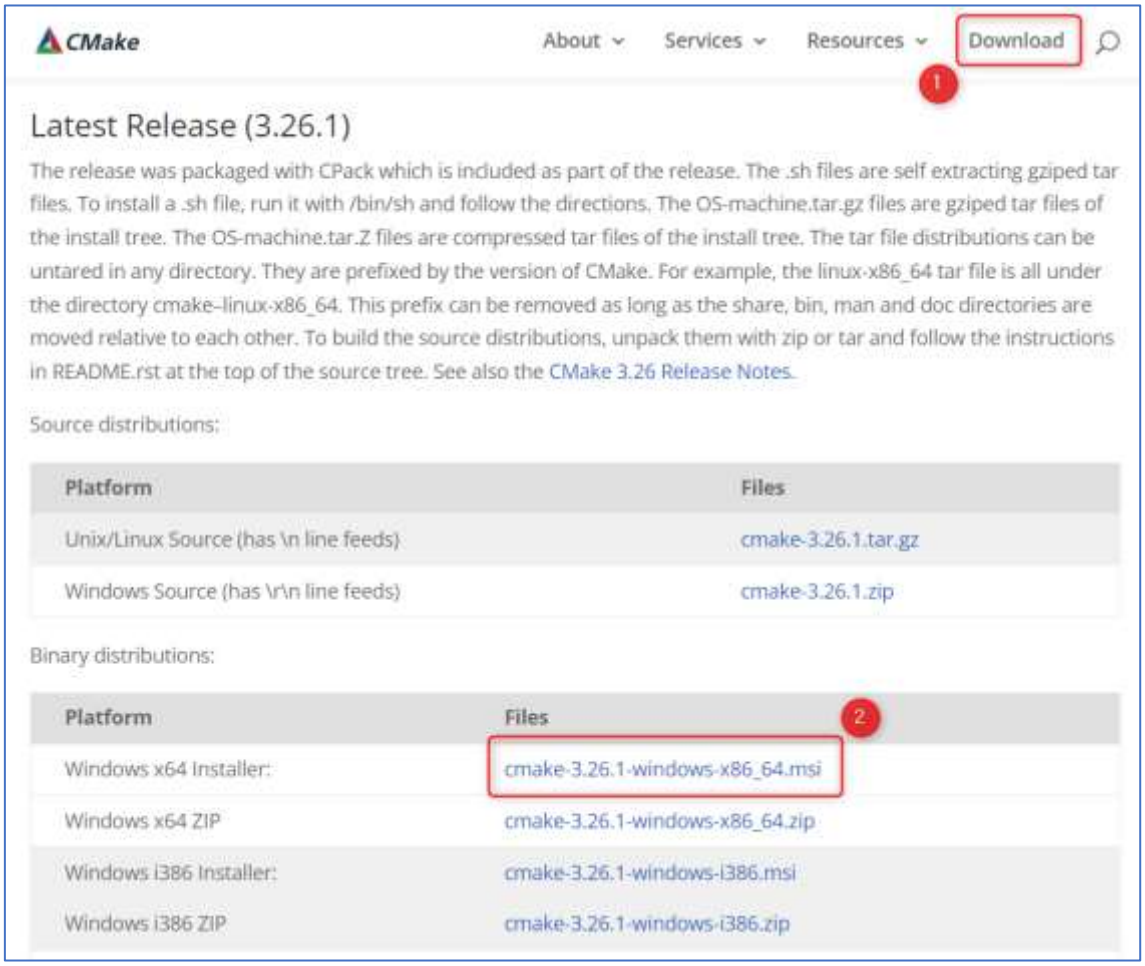
C/C++は非常に自由度が高く何でもできる反面、ちょっとしたミスですぐに不具合を起こします。このためソースコードレベルでテストを行うことが必須となります。これを行うのが単体テストです。

2. 必要なもの

2.1. CMake のダウンロード

(1) 「https://cmake.org/」にアクセスします。

(2) 下図①②の順に操作し、msi ファイルをダウンロードします。



2.2. eclipse のダウンロード

(1) 「https://mergedoc.osdn.jp/」にアクセスします。

(2) 最新年のボタンをクリックします。

(3) 使用する言語に合わせたボタンをクリックします。ここで必要なのは C/C++のみですが、複数言語を使用する場合は Ultimate をどうぞ。

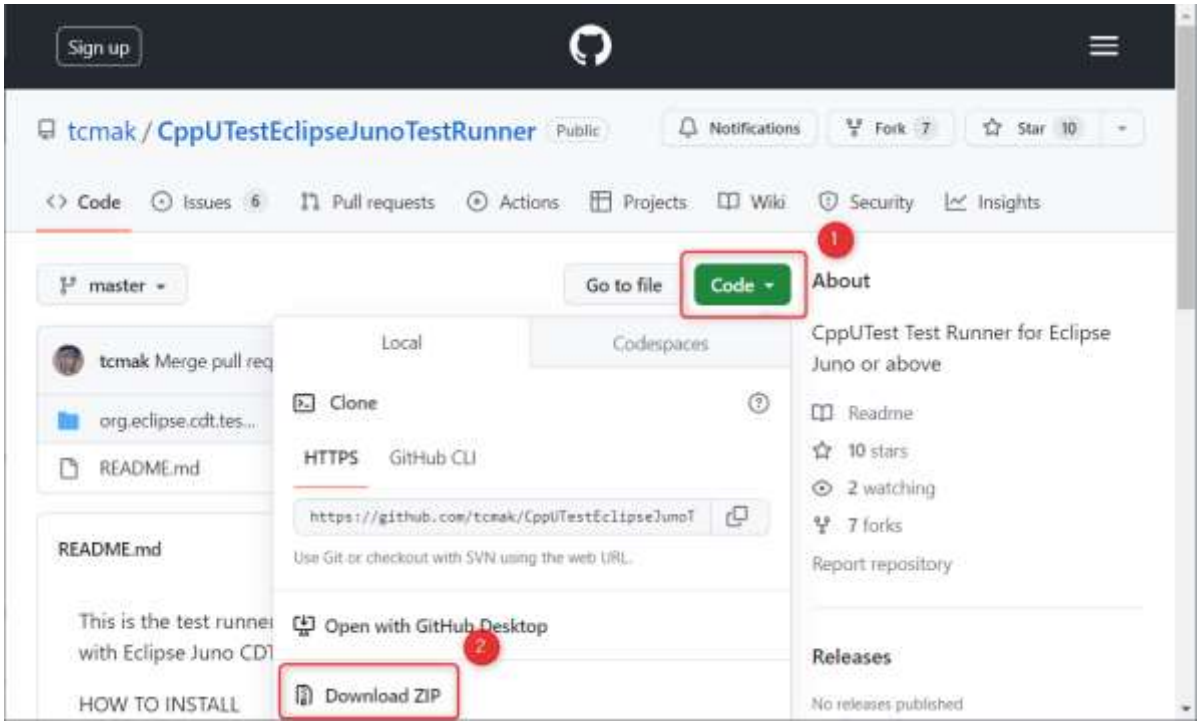
(4) リンクをクリックして exe をダウンロードします。

2.3. CppUtest Test Runner のダウンロード

eclipse で CppUTest を使用するためのプラグインです。

(1) 「https://github.com/tcmak/CppUTestEclipseJunoTestRunner」にアクセスします。

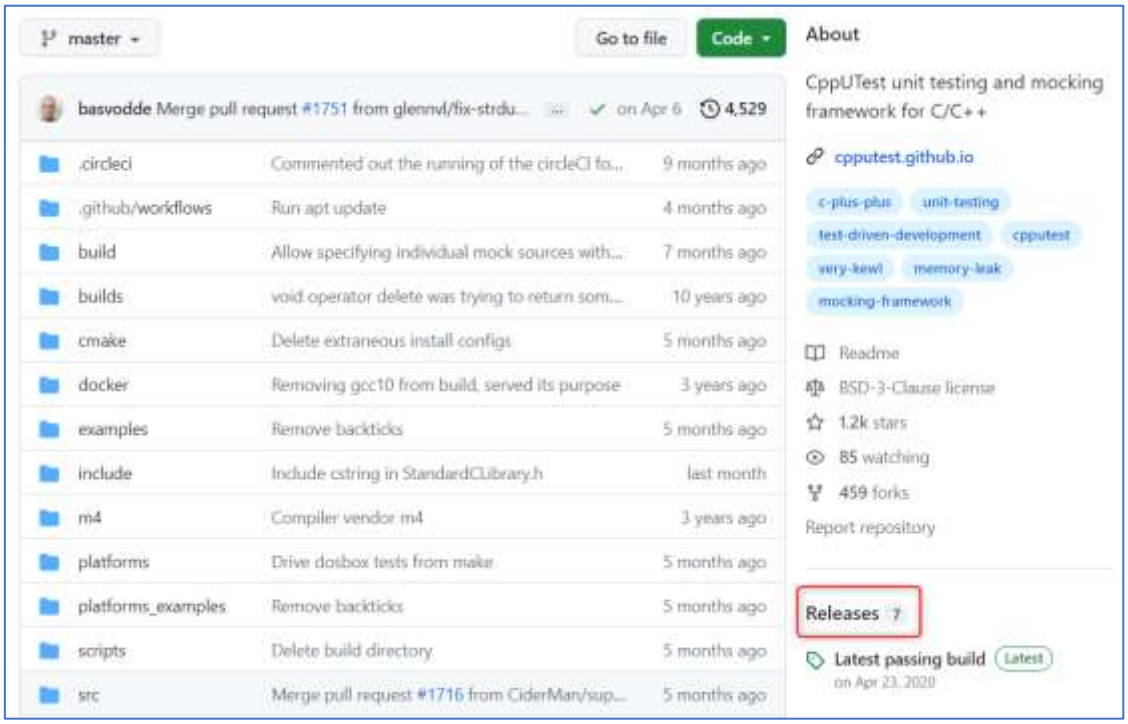
(2) 下図①②の順に操作し、zip ファイルをダウンロードします。



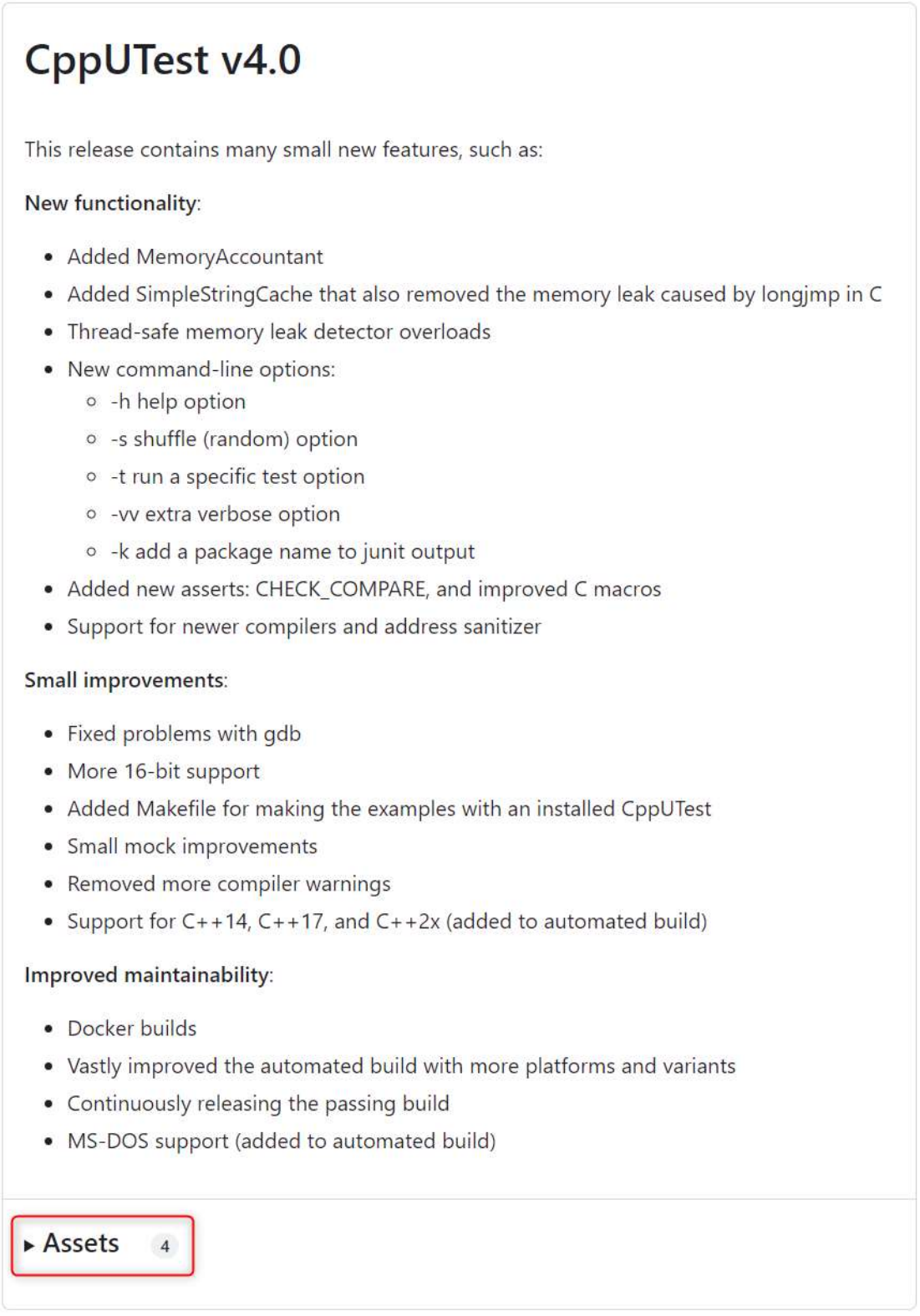
2.4. CppUTest のダウンロード

(1) 「https://github.com/cpputest/cpputest」にアクセスします。

(2) 「Releases」リンクをクリックします。



(3) 「Assets」をクリックします。



CppUTest v4.0

This release contains many small new features, such as:

New functionality:

- Added MemoryAccountant
- Added SimpleStringCache that also removed the memory leak caused by longjmp in C
- Thread-safe memory leak detector overloads
- New command-line options:
  - -h help option
  - -s shuffle (random) option
  - -t run a specific test option
  - -vv extra verbose option
  - -k add a package name to junit output
- Added new asserts: CHECK_COMPARE, and improved C macros
- Support for newer compilers and address sanitizer

Small improvements:

- Fixed problems with gdb
- More 16-bit support
- Added Makefile for making the examples with an installed CppUTest
- Small mock improvements
- Removed more compiler warnings
- Support for C++14, C++17, and C++2x (added to automated build)

Improved maintainability:

- Docker builds
- Vastly improved the automated build with more platforms and variants
- Continuously releasing the passing build
- MS-DOS support (added to automated build)

▸ Assets 4

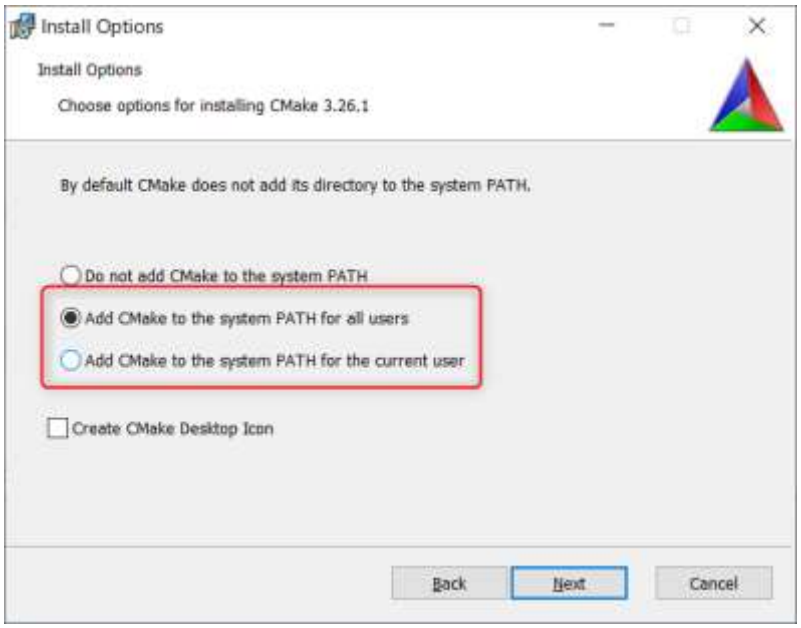
(4) zip ファイルをダウンロードします。

3. インストール

3.1. CMake のインストール

(1) ダウンロードした msi ファイルをダブルクリックして、インストールを開始します。

(2) インストールは基本的に [Next] ボタンをクリックするだけですが、下記はいずれかを選択してください。

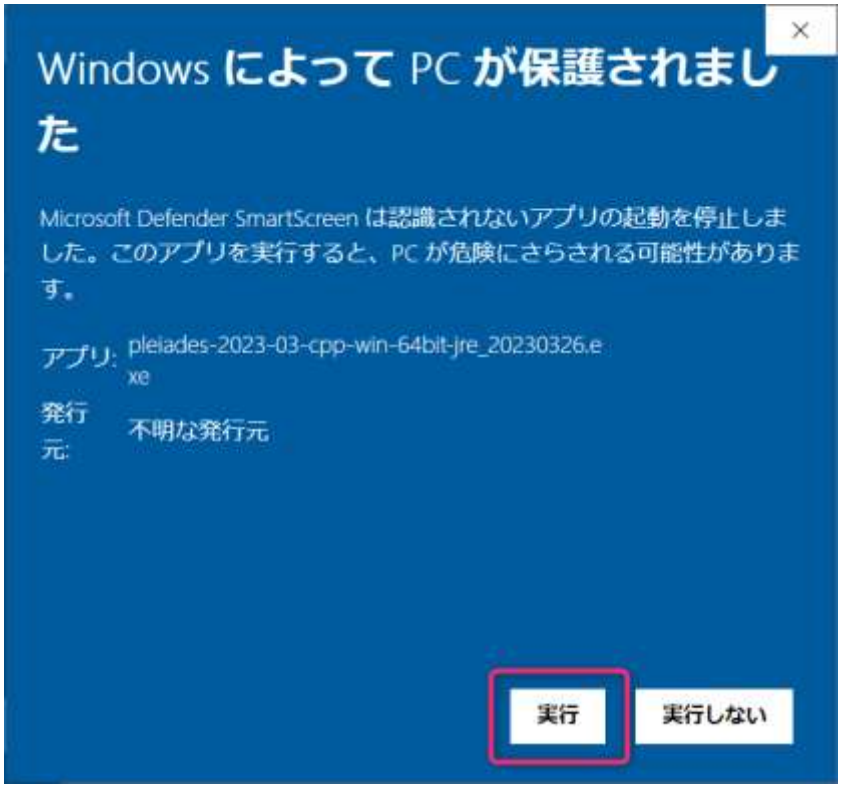


3.2. eclipse のインストール

(1) ダウンロードした exe ファイルをダブルクリックして、インストールを開始します。

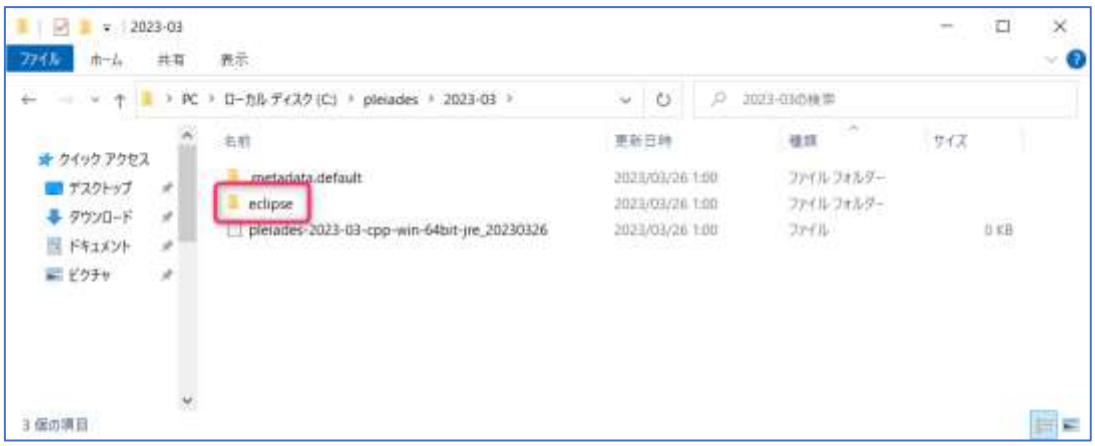
(2) 下図の警告のようなウィンドウが表示された場合、[詳細情報]をクリックします。

(3) [実行]ボタンをクリックします。

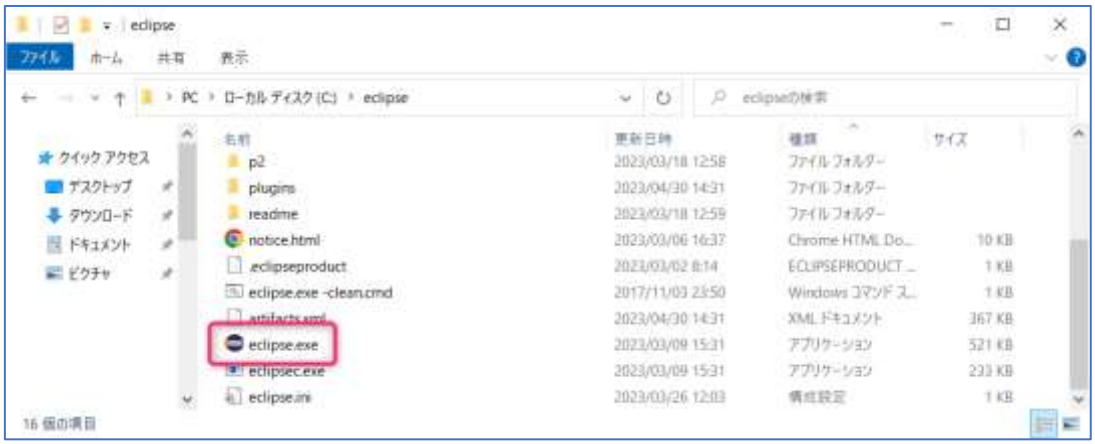


(4) [解凍]ボタンをクリックします。

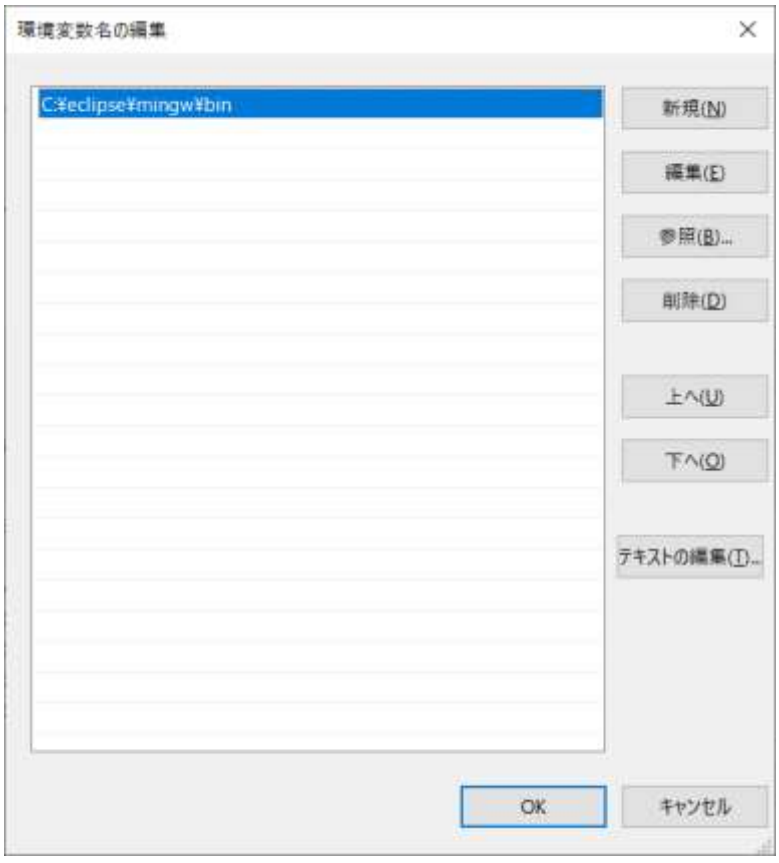
(5) 解凍したフォルダーの中にある、eclipse フォルダーを C ドライブの直下に移動します。



(6) 「eclipse.exe」を右クリックして、スタートメニューにピン留めしておくと便利です。



(7) 環境変数の Path に「C:¥eclipse¥mingw¥bin」を追加します。

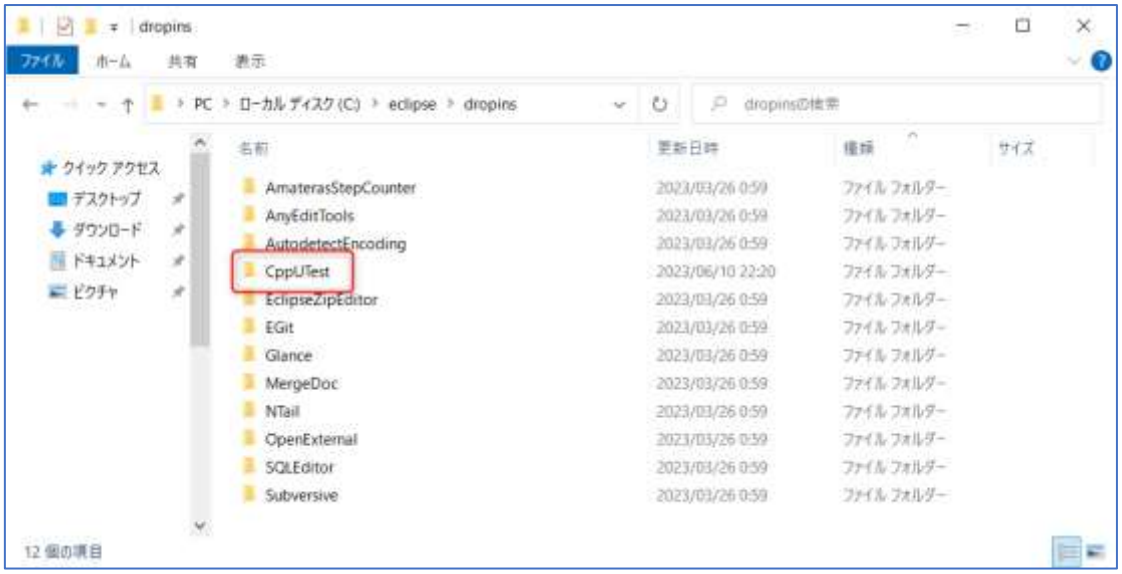


(8) Windows を再起動します。

3.3. CppUTest Test Runner のインストール

(1) 「C:¥eclipse¥dropins」を開きます。

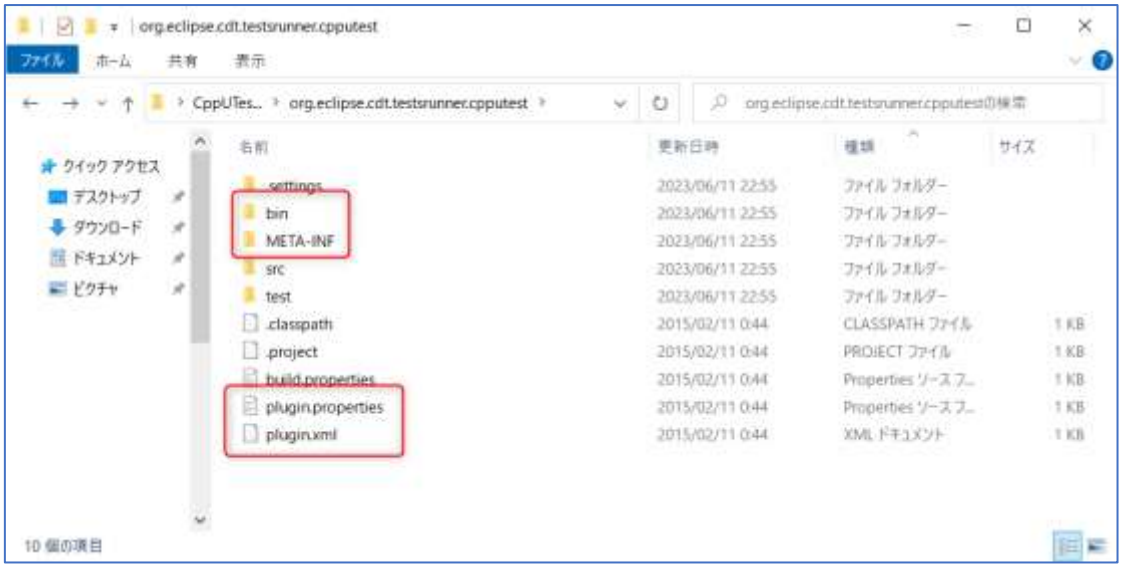
(2) 「CppUTest」フォルダーを作成します。



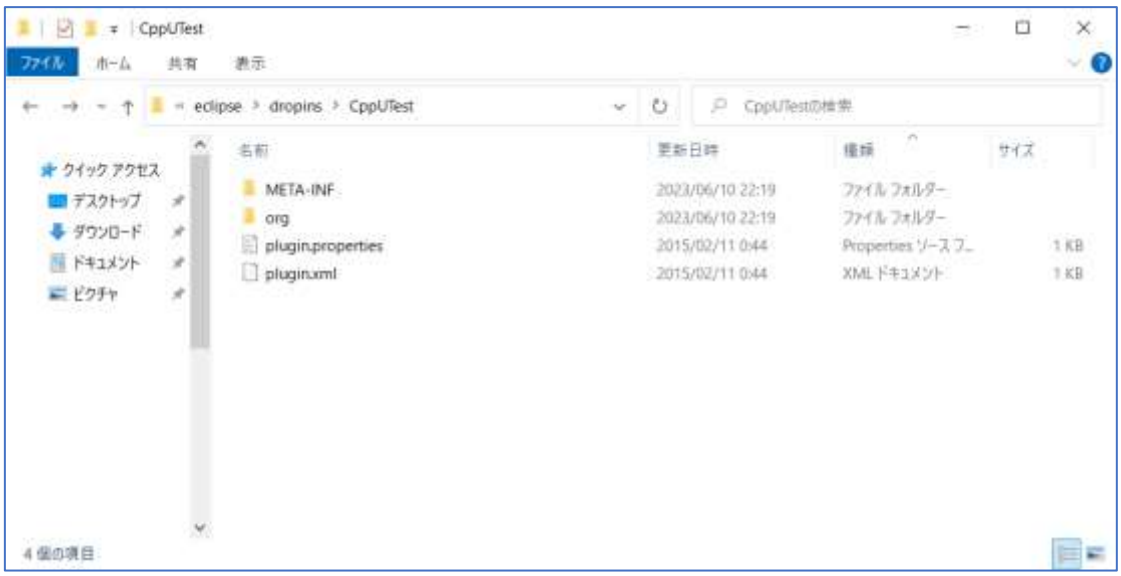
(3) ダウンロードした zip ファイルを解凍し、「org.eclipse.cdt.testsrunner.cpputest」フォルダーを開きます。

(4) 「org.eclipse.cdt.testsrunner.cpputest」フォルダー内にある下記のフォルダーやファイルを、「C:¥eclipse¥dropins¥CppUTest」フォルダーに移動します。

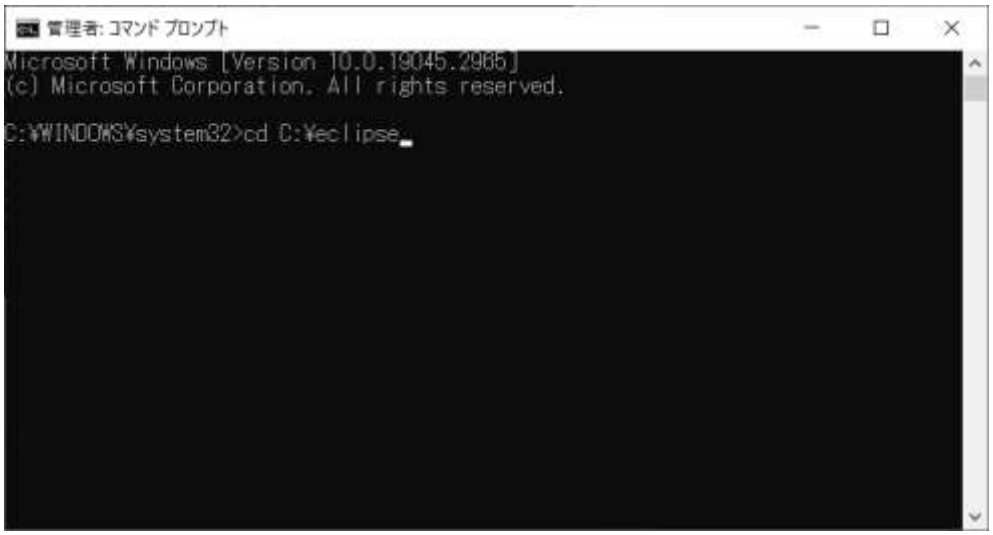
・bin フォルダー内にある org フォルダー
・META-INF フォルダー
・plugin.properties ファイル
・plugin.xml ファイル



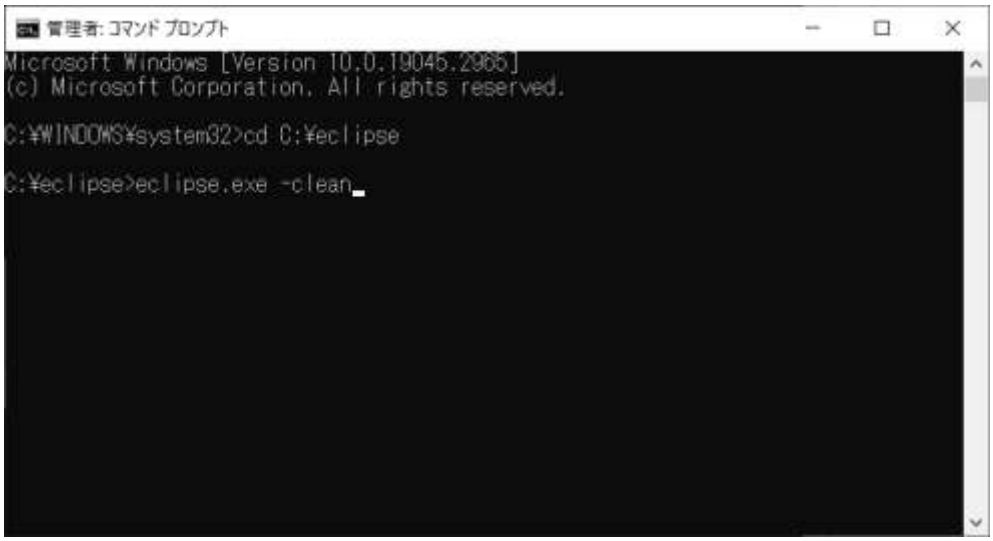
(5) 移動後のフォルダー内の構成は下図のとおりになります。



(6) コマンドプロンプトを起動し、「C:¥eclipse」フォルダーに移動します。



(7) 「eclipse.exe -clean」を実行します。



3.4. CppUTest のインストール

(1) ダウンロードした zip ファイルを解凍します。

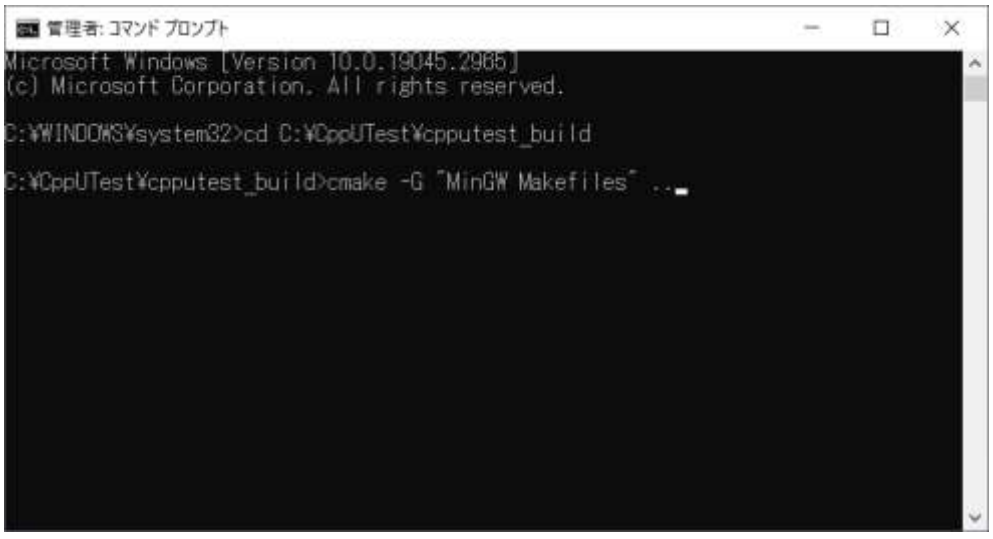
(2) 解凍してできたフォルダー名を「CppUTest」に変更します。

(3) CppUTest フォルダーを C ドライブの直下に置きます。

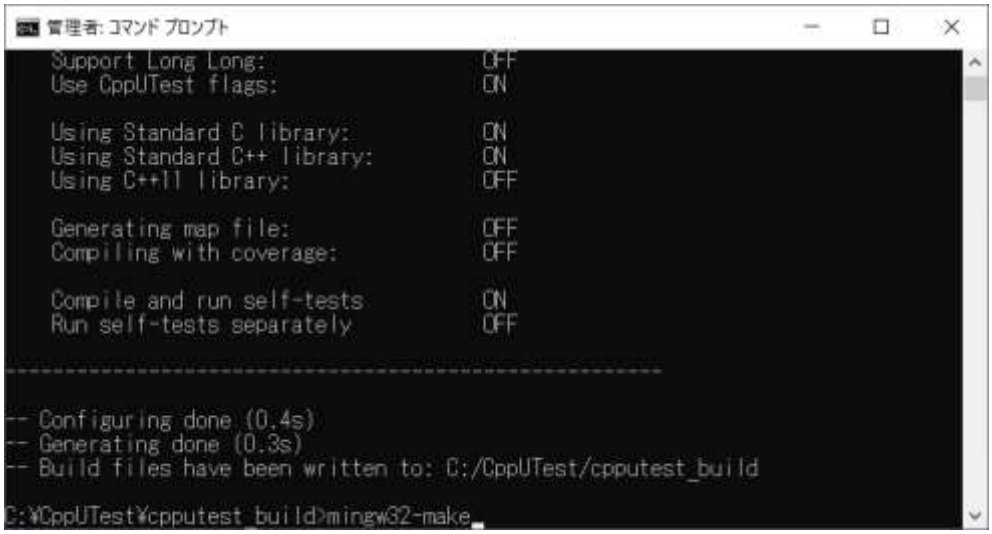
(4) スタートメニューから①コマンドプロンプトを探し出して右クリックし、「管理者として実行」をクリックします。

(5) 「cd C:¥CppUTest¥cpputest_build」と入力して Enter キーを押します。

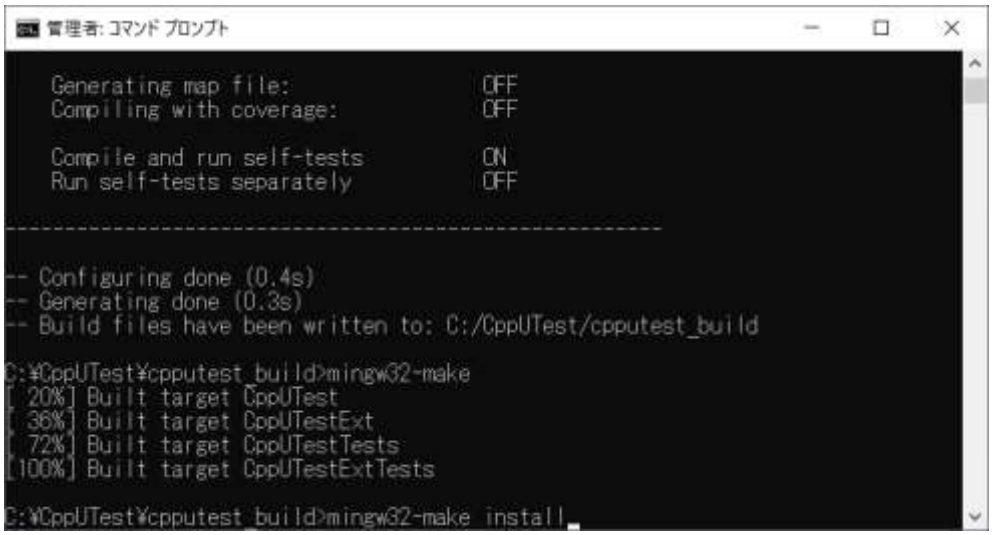
(6) 「cmake -G "MinGW Makefiles" ..」と入力して Enter キーを押します。



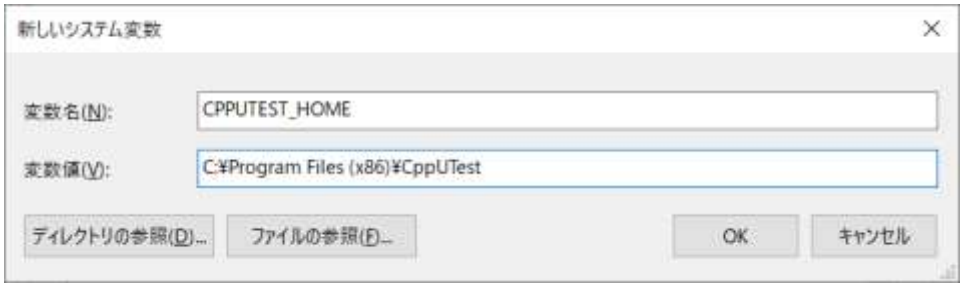
(7) 「mingw32-make」と入力して Enter キーを押します。



(8) 「mingw32-make install」と入力して Enter キーを押します。



(9) 環境変数に新規で「CPPUTEST_HOME」「C:¥Program Files (x86)¥CppUTest」を追加します。

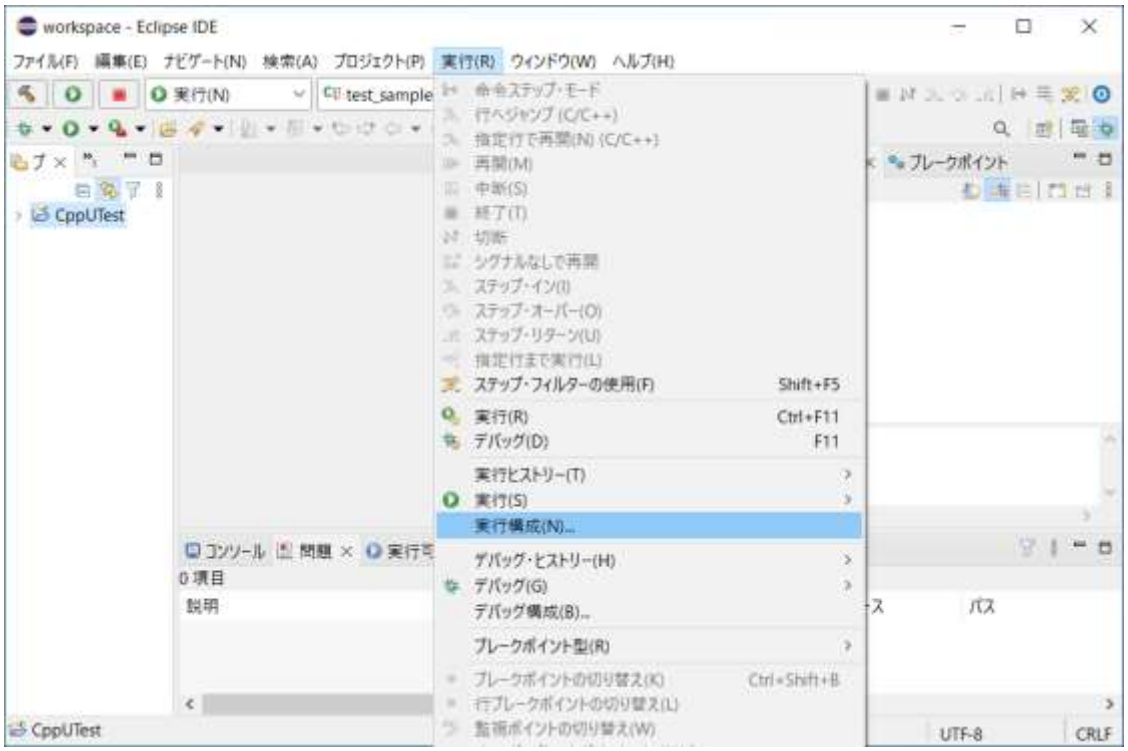


(10) Windows を再起動します。

4. eclipse の設定

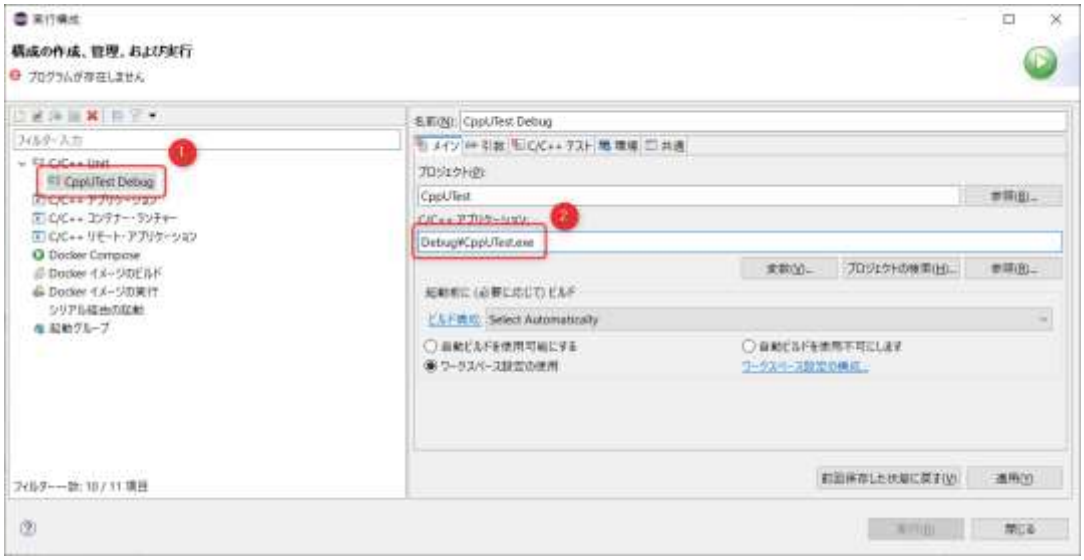
4.1. 実行構成の設定

(1) メニュー [実行 > 実行構成] をクリックします。

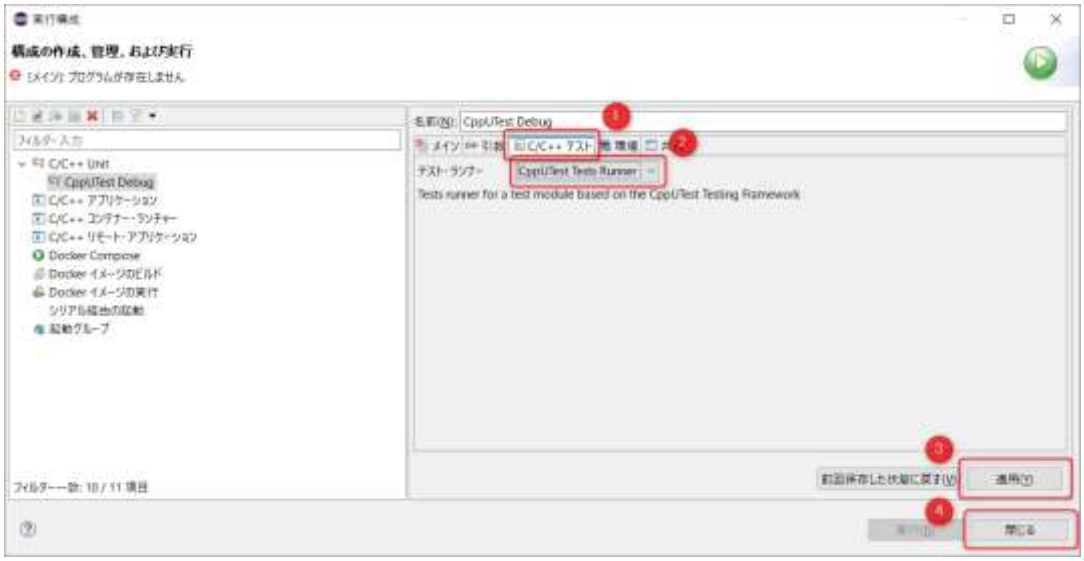


(2) ①をダブルクリックします。

(3) ①が表示されるので選択し、②に「Debug¥CppUTest.exe」と入力します。

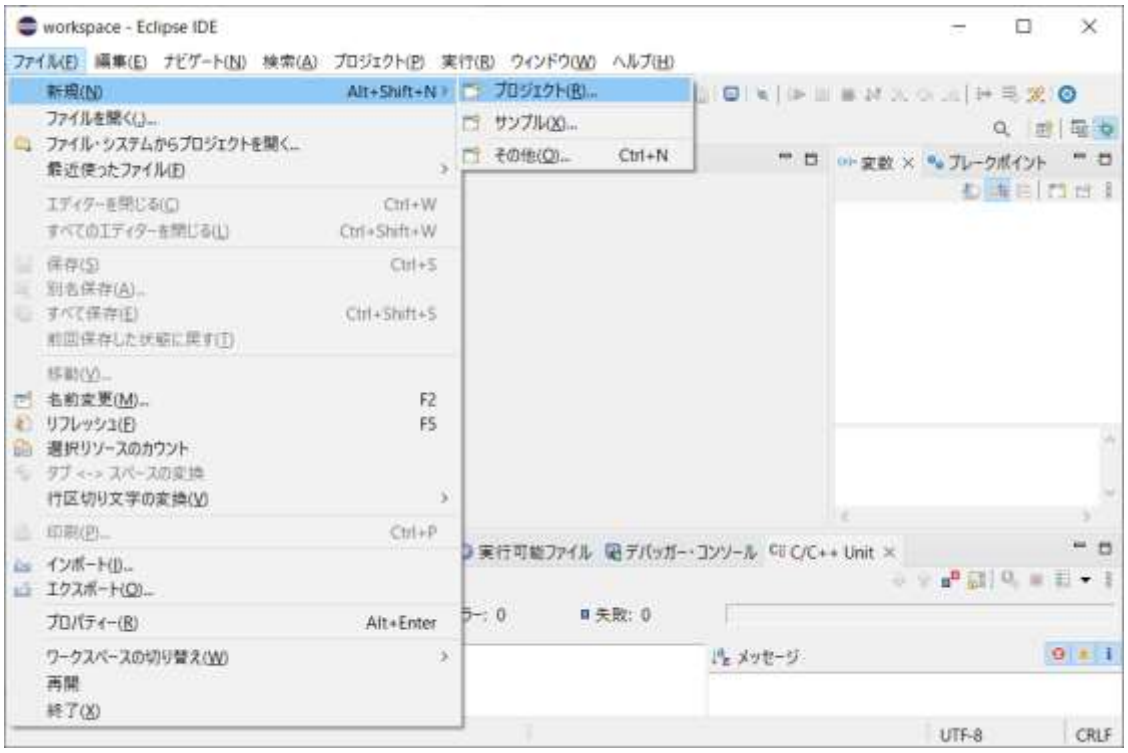


(4) 下図①を選択した後、②に「CppUTest Tests Runner」を選択します。最後に③④の順にボタンをクリックして終了です。



5. プロジェクトの作成

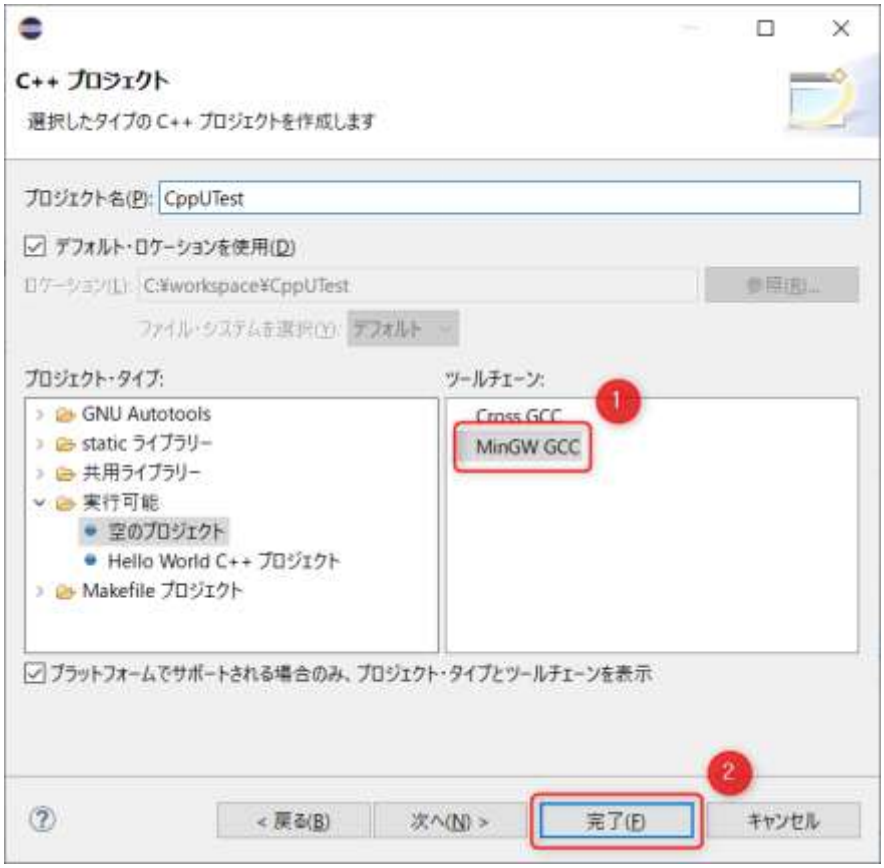
(1) eclipse を起動し、メニュー [ファイル > 新規 > プロジェクト] をクリックします。



(2) 下図①を選択した後、② [次へ] ボタンをクリックします。

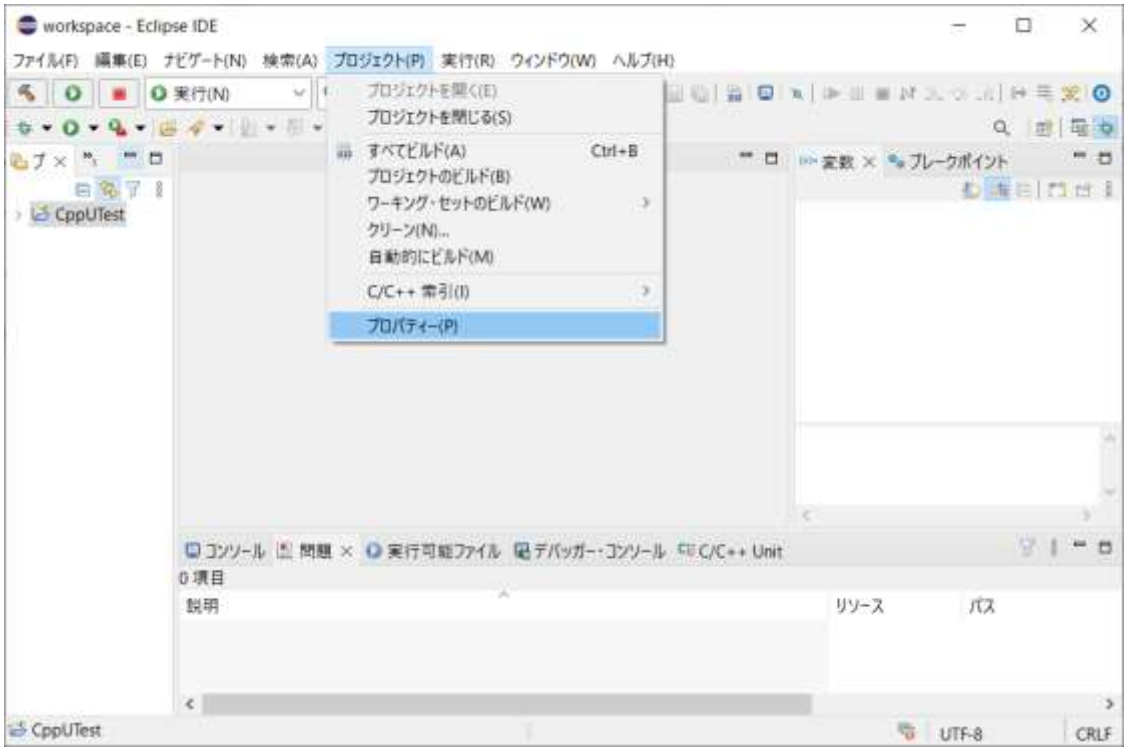
(3) 下図①を選択した後、② [次へ] ボタンをクリックします。

(4) 下図①を選択した後、② [完了] ボタンをクリックします。



5.1. プロジェクトの設定

(1) メニュー [プロジェクト > プロパティー] をクリックします。



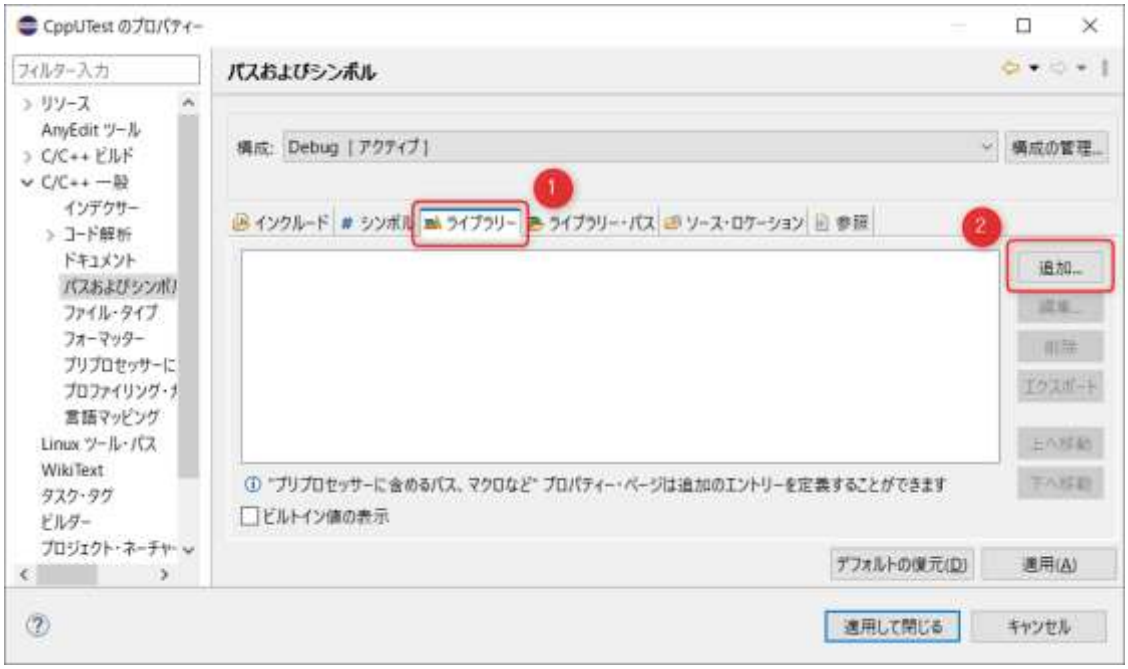
(2) 下図①～③の順に選択した後、④ [追加] ボタンをクリックします。

(3) 下図①に「${CPPUTEST_HOME}/include」を入力した後、② [OK] ボタンをクリックします。

(4) 続けてもう一度 [追加] ボタンをクリックします。

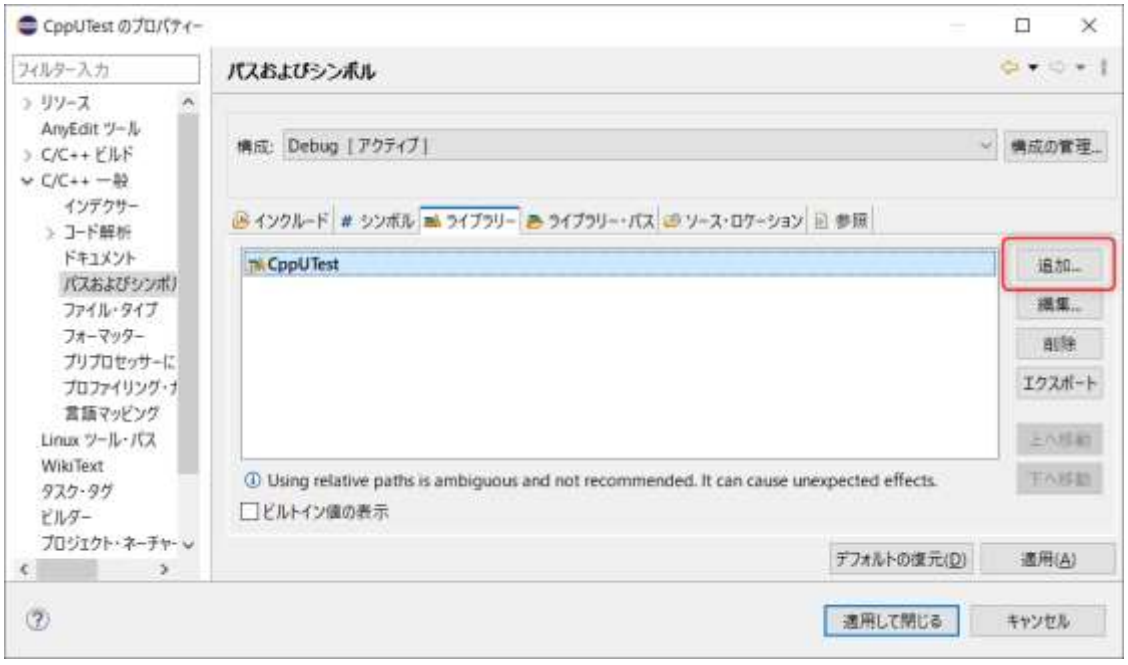
(5) 下図①に「${CPPUTEST_HOME}/include/CppUTest」を入力した後、② [OK] ボタンをクリックします。

(6) 下図①を選択した後、② [追加] ボタンをクリックします。



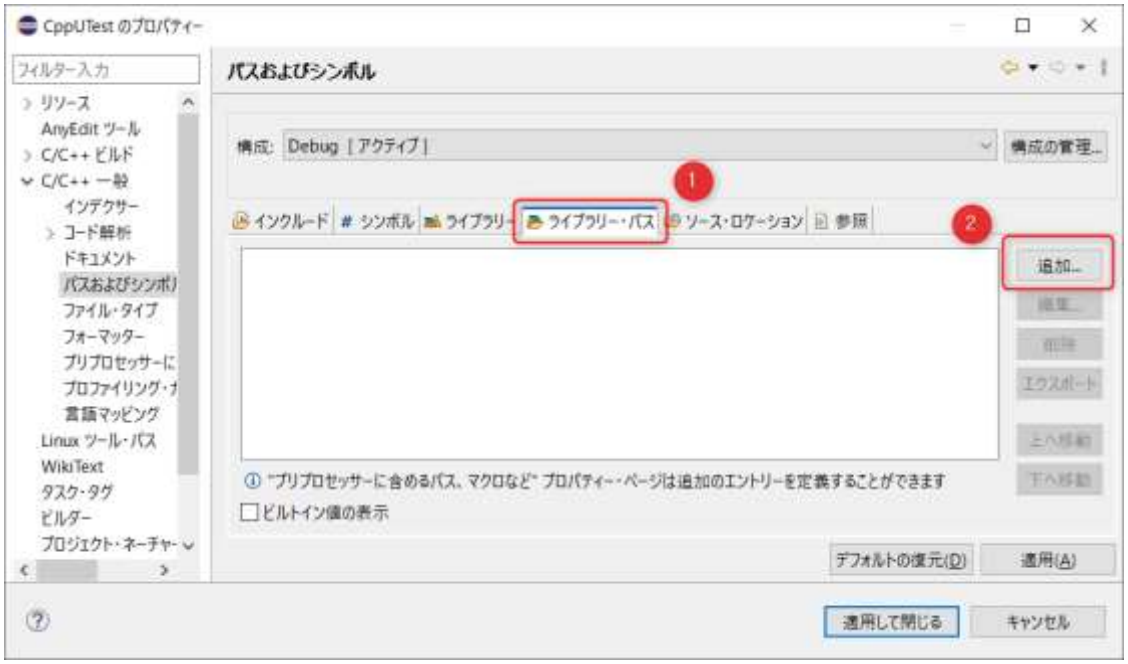
(7) 下図①に「CppUTest」を入力した後、② [OK] ボタンをクリックします。

(8) 続けてもう一度 [追加] ボタンをクリックします。



(9) 下図①に「CppUTestExt」を入力した後、② [OK] ボタンをクリックします。

(10) 下図①を選択した後、② [追加] ボタンをクリックします。



(11) 下図①に「${CPPUTEST_HOME}/lib」を入力した後、② [OK] ボタンをクリックします。

(12) [適用して閉じる] ボタンをクリックします。

5.2. プロジェクトの構成例

下記のようなプロジェクト構成であったとします。

```
C:¥workspace
└ CppUTest
   └ src
      ├ AllTests.cpp      ... テストコード
      ├ base64Test.cpp    ... テストコード
      ├ base64.c    ... テスト対象
      └ base64.h    ... テスト対象
```

base64.c：テスト対象のソースコードです。

今回の例では「base64_encode」関数が正しく動作しているかをテストします。

```
//! Base64 エンコードテーブル
static const uint8_t m_enc_table[] =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

/** ------------------------------------------------------------------------
 * @brief Base64 エンコード
 *
 * @param [in] bin : バイナリーデータ
 * @param bin_size : バイナリーデータ長
 * @param [out] ascii : ASCII データ
 * @return ASCII データ長
 */
int32_t base64_encode(const uint8_t bin[], uint32_t bin_size, uint8_t ascii[])
{
  uint8_t  bit6;
  uint8_t  buf_size   = 0;
  uint16_t buf        = 0;
  uint32_t ascii_size = 0;

  for (uint32_t i = 0; i < bin_size; i++) {
```

```
    buf = (uint16_t)(buf << 8) | bin[i];
    buf_size += 8;

    // 6 ビット毎に変換
    do {
      bit6 = (uint8_t)(buf >> (buf_size - 6)) & 0x3f;
      buf_size -= 6;
      ascii[ascii_size] = m_enc_table[bit6];
      ascii_size++;
    } while (buf_size >= 6);
  }

  // 端数ビットがある場合は 0 を補完して変換
  if (buf_size > 0) {
    bit6              = (uint8_t)(buf << (6 - buf_size)) & 0x3f;
    ascii[ascii_size] = m_enc_table[bit6];
    ascii_size++;
  }

  // バイト数が 4 の倍数でないなら'='を追加
  while ((ascii_size % 4) != 0) {
    ascii[ascii_size] = (uint8_t)'=';
    ascii_size++;
  }
  ascii[ascii_size] = (uint8_t)'¥0';

  return (int32_t)ascii_size;
}
```

AllTests.cpp：何も考えずに、このファイルを作成しておいてください。

```
#include "CppUTest/CommandLineTestRunner.h"

int main(int argc, char *argv[])
{
  return CommandLineTestRunner::RunAllTests(argc, argv);
}
```

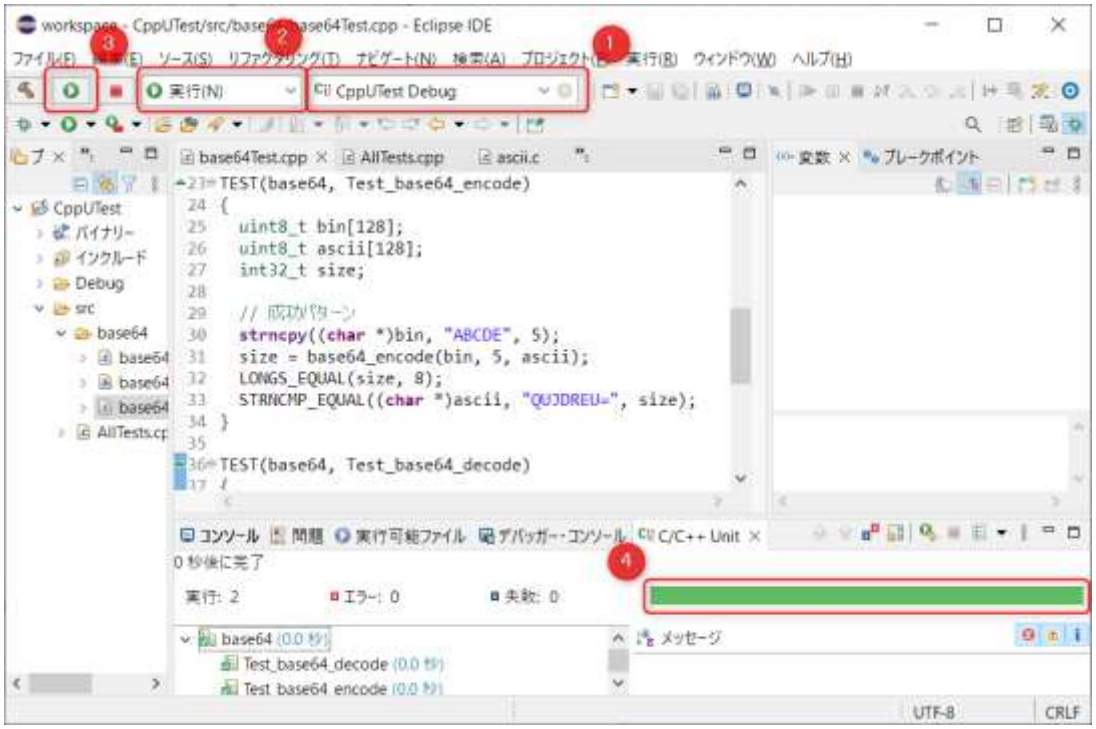
base64Test.cpp：「base64.c」ファイル用のテストコードです。
ファイル単位でテストコードを用意すると良いでしょう。

```
#include "CppUTest/TestHarness.h"    // 必要なもの

#include <string.h>
#include "base64.h"

// テストグループとテストグループ内の各テストを実行する前後に行う処理を定義します
// ( )内の"base64"は識別子で、テスト対象のファイル名にすると良いでしょう
TEST_GROUP(base64){
  // 各テストケースの実行直前に呼ばれる仮想メソッド
  TEST_SETUP(){
  }

  // 各テストケースの実行直後に呼ばれる仮想メソッド
  TEST_TEARDOWN(){
  }
};

// 実行するテストを記述します
// "base64"は TEST_GROUP と同じ識別子で、"Test_base64_encode"はテストの識別子です
```

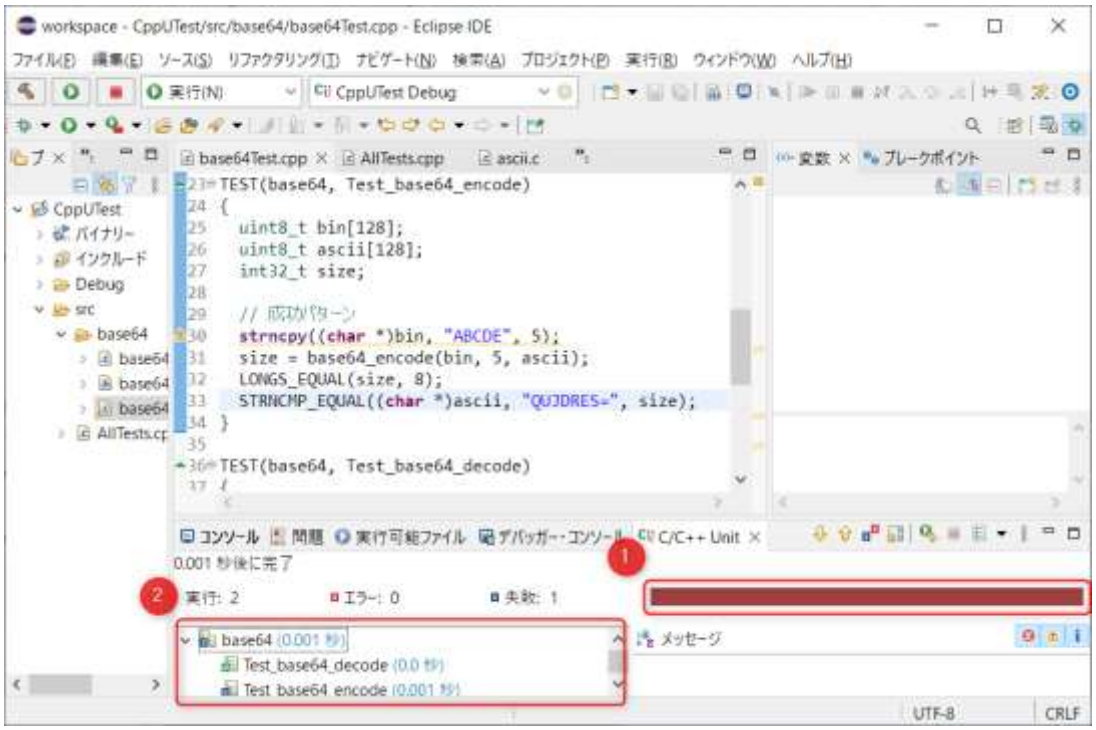
```
// テストの識別子
TEST(base64, Test_base64_encode)
{
  uint8_t bin[128];
  uint8_t ascii[128];
  int32_t size;

  strncpy((char *)bin, "ABCDE", 5);
  size = base64_encode(bin, 5, ascii);
  LONGS_EQUAL(size, 8);    // テスト 1
  STRNCMP_EQUAL((char *)ascii, "QUJDREU=", size);    // テスト 2
}

// 他の関数のテストも行うなら、TEST を分けると良いでしょう
TEST(base64, Test_base64_decode)
{
  ...
}
```

5.3. プロジェクトの実行

(1) ①で CppUTest を、②で実行を選択し、③のボタンをクリックします。その結果すべてのテストに合格すると、④に緑色のバーが表示されます。



(2) 1つでもテストで誤りが見つかった場合、①に赤色のバーが表示されます。
また②にはテストの一覧が表示されています。この中で誤りの発生したテストは、アイコンの左下に青色で小さく「×」と表示されます。そこをダブルクリックしていくとテストに引っ掛かった箇所へ移動できるので、誤りを訂正してください。



6. TEST マクロ

TEST マクロ	内容
CHECK(boolean condition)	ブール結果が true かをチェックします。
CHECK_FALSE(condition)	ブール結果が false かをチェックします。
STRCMP_EQUAL(expected, actual)	strcmp を使用して、文字列が等しいかをチェックします。
STRNCMP_EQUAL(expected, actual, length)	strncmp を使用して、文字列が等しいかをチェックします。
STRCMP_NOCASE_EQUAL(expected, actual)	大文字と小文字を区別せずに、文字列が等しいかをチェックします。
STRCMP_CONTAINS(expected, actual)	actual に expected が含まれているかをチェックします。
LONGS_EQUAL(expected, actual)	2 つの数値を比較します。
UNSIGNED_LONGS_EQUAL(expected, actual)	2 つの正の数を比較します。
LONGLONGS_EQUAL(expected, actual)	2 つの数値を比較します。
UNSIGNED_LONGLONGS_EQUAL(expected, actual)	2 つの正の数を比較します。
BYTES_EQUAL(expected, actual)	8 ビット幅の 2 つの数値を比較します。
POINTERS_EQUAL(expected, actual)	2 つのポインターを比較します。
DOUBLES_EQUAL(expected, actual, tolerance)	2 つの浮動小数点数が許容範囲内（tolerance）かを比較します。
FUNCTIONPOINTERS_EQUAL(expected, actual)	2 つの void(*)関数ポインターを比較します。
MEMCMP_EQUAL(expected, actual, size)	メモリーの 2 つの領域を比較します。
BITS_EQUAL(expected, actual, mask)	マスクを適用して、期待値と実際のビットを少しずつ比較します。

```
CHECK(true);            // 判定値が true なら合格
CHECK_FALSE(false);     // 判定値が false なら合格

STRCMP_EQUAL("abc", "abc");          // 文字列が一致すれば合格
STRNCMP_EQUAL("abc1", "abc2", 3);    // 文字列が先頭から指定文字数一致すれば合格
STRCMP_NOCASE_EQUAL("abc", "ABC");   // 大文字・小文字を区別せず一致すれば合格
STRCMP_CONTAINS("123", "ABC123");    // 文字列が含まれていれば合格

// 数値が一致すれば合格
LONGS_EQUAL(-1, -1);
UNSIGNED_LONGS_EQUAL(1, 1);
LONGLONGS_EQUAL(-1, -1);
UNSIGNED_LONGLONGS_EQUAL(1, 1);
BYTES_EQUAL(-1, 255);
DOUBLES_EQUAL(1.000, 1.001, 0.001);
BITS_EQUAL(0x31, 0x01, 0x0F);

MEMCMP_EQUAL("string", "string", 6);

// ポインターが一致すれば合格
POINTERS_EQUAL(0, 0);
FUNCTIONPOINTERS_EQUAL(0, 0);
```

7. CppUMock

入力に対して計算を行い、出力を返すような関数であればテストは簡単です。しかしハードに依存する部分のテストは容易ではありません。たとえばマイコンの GPIO 制御です。次のようなスイッチ入力を判定する処理があるとします。

```
/** ----------------------------------------------------------------------------
 * @brief キーの確認
 * @retval 0 : オフ
 * @retval 1 : オン
 */
uint8_t Check_Key(void)
{
  uint8_t key;

  key = GPIOA.IDR & KEY1;
  if (key == 0) {
    return 1;
  } else {
    return 0;
  }
}
```

上記例では GPIO の制御がハード依存のため、このままではテストできません。そこでまずは、ハード依存の部分を関数化して隠します。HAL（Hardware Abstraction Layer：ハードウェア抽象化レイヤー）が用意されているならそれを使ってください。なければ似たような関数を自作して置き換えます。

```
key = GPIO_ReadPin(GPIOA, KEY1);
```

これで準備は完了です。

7.1. CppUMock の使用方法

CppUTest には、CppUMock というモック（模造などの意味）機能があります。このモック機能を用いることで、本来の関数を模造したダミー関数を作ることができます。今回の例でいうと GPIO_ReadPin のモック関数を用意し、テストするのに都合の良い応答を返すようにします。
CppUMock は C と C++で記述方法が少し変わります。以降の解説は C をベースに行い、『7.2. サンプル』にて C と C++の比較を掲載します。記述方法の違いについては、そちらで確認してください。

さて、さっそくですが GPIO_ReadPin のモック関数が次のようになったとします。

```
uint8_t GPIO_ReadPin(uint16_t port, uint8_t pin)
{
  mock_c()
      ->actualCall("GPIO_ReadPin")
      ->withUnsignedIntParameters("port", port)
      ->withUnsignedIntParameters("pin", pin);
  return mock_c()->intReturnValue();
}
```

表 1　モック関数の意味

`actualCall("GPIO_ReadPin")`	GPIO_ReadPin という関数です。
`withUnsignedIntParameters("port", port)`	1 つ目の引数は unsigned 型で、その名前は「port」です。引数の宣言は型によって変わります。詳細は『7.3. 引数』を参照ください。
`withUnsignedIntParameters("pin", pin)`	2 つ目の引数は unsigned int 型で、その名前は「pin」です。
`intReturnValue()`	戻り値は int 型です。戻り値の宣言は型によって変わります。詳細は『7.4. 戻り値』を参照ください。

このモック関数に期待した動きをさせるには、次のように記述します。

```
mock_c()
    ->expectOneCall("GPIO_ReadPin")
    ->withUnsignedIntParameters("port", GPIOA)
    ->withUnsignedIntParameters("pin", KEY1)
    ->andReturnIntValue(0);
```

表 2　期待するモック関数の呼び出し記録の意味

`expectOneCall("GPIO_ReadPin")`	GPIO_ReadPin 関数が 1 度呼び出されたときの期待する効果を記録します。
`withUnsignedIntParameters("port", GPIOA)`	1 つ目の引数「port」は unsigned int 型で、GPIOA を渡します。引数の宣言は型によって変わります。詳細は『7.3. 引数』を参照ください。 なお引数としてどの様な値を渡すかは重要ではありません。
`withUnsignedIntParameters("pin", KEY1)`	2 つ目の引数「pin」は unsigned int 型で、KEY1 を渡します。
`andReturnIntValue(0)`	関数の戻り値として「0」を返します。

たとえばチャタリング対策で 3 回連続で同じ値なら採用する、というテストを行う場合は、3 回記述します。

```
mock_c()
    ->expectOneCall("GPIO_ReadPin")
    ->withUnsignedIntParameters("port", GPIOA)
    ->withUnsignedIntParameters("pin", KEY1)
    ->andReturnIntValue(0);
mock_c()
    ->expectOneCall("GPIO_ReadPin")
    ->withUnsignedIntParameters("port", GPIOA)
    ->withUnsignedIntParameters("pin", KEY1)
    ->andReturnIntValue(0);
mock_c()
    ->expectOneCall("GPIO_ReadPin")
    ->withUnsignedIntParameters("port", GPIOA)
    ->withUnsignedIntParameters("pin", KEY1)
    ->andReturnIntValue(0);
```

または「expectNCalls」を用いることで、指定回数分の挙動を一度に設定することもできます。

```
mock_c()
    ->expectNCalls(3, "GPIO_ReadPin")  // 期待した効果を 3 回分記録する
    ->withUnsignedIntParameters("port", GPIOA)
    ->withUnsignedIntParameters("pin", KEY1)
    ->andReturnIntValue(0);
```

7.2. サンプルコード

先のテストコードの全体を示します。

7.2.1. テスト対象コード

key.c：テスト対象

```
//-----------------------------------------------------------------------------
// include
//-----------------------------------------------------------------------------
#include "gpio.h"
#include "key.h"

/** ----------------------------------------------------------------------------
 * @brief キーの確認
 * @retval 0 : オフ
 * @retval 1 : オン
 */
uint8_t Check_Key(void)
{
  uint8_t key;

  key = GPIO_ReadPin(GPIOA, KEY1);
  if (key == 0) {
    return 1;
  } else {
    return 0;
  }
}
```

key.h：テスト対象

```
#ifndef KEYH
#define KEYH

#ifdef __cplusplus
extern "C" {
#endif

//-----------------------------------------------------------------------------
// include
//-----------------------------------------------------------------------------
#include <stdint.h>

//-----------------------------------------------------------------------------
// function
//-----------------------------------------------------------------------------
```

```
uint8_t Check_Key(void);

#ifdef __cplusplus
}
#endif

#endif
```

gpio.h：GPIO の HAL サンプル

```
#ifndef GPIOH
#define GPIOH

#ifdef __cplusplus
extern "C" {
#endif

//-------------------------------------------------------------------------------
// include
//-------------------------------------------------------------------------------
#include <stdint.h>

//-------------------------------------------------------------------------------
// define
//-------------------------------------------------------------------------------
#define KEY1 (0x01)

//-------------------------------------------------------------------------------
// function
//-------------------------------------------------------------------------------
uint8_t GPIO_ReadPin(uint16_t port, uint8_t pin);

#ifdef __cplusplus
}
#endif

#endif
```

7.2.2. C のテストコード

C で記述した場合のテストコードです。

keyTest.cpp：モックを用いた C のテストコード

```
#include "CppUTest/TestHarness.h"
#include "CppUTestExt/MockSupport_c.h"

#include "gpio.h"
#include "key.h"

// clang-format off
TEST_GROUP(key){
  TEST_SETUP(){
  }

  TEST_TEARDOWN(){
    mock_c()->clear();  // モック資源のクリアー
  }
```

```
};
// clang-format on

TEST(key, Test_Mock)
{
  uint8_t key;

  mock_c()
      ->expectOneCall("GPIO_ReadPin")
      ->withUnsignedIntParameters("port", GPIOA)
      ->withUnsignedIntParameters("pin", KEY1)
      ->andReturnIntValue(0);

  key = Check_Key();
  LONGS_EQUAL(key, 1);

  // 結果検証
  mock_c()->checkExpectations();
}

/** ---------------------------------------------------------------------------
 * @brief モック関数
 */
uint8_t GPIO_ReadPin(uint16_t port, uint8_t pin)
{
  mock_c()
      ->actualCall("GPIO_ReadPin")
      ->withUnsignedIntParameters("port", port)
      ->withUnsignedIntParameters("pin", pin);
  return mock_c()->intReturnValue();
}
```

7.2.3. C++のテストコード

C++で記述した場合のテストコードです。

keyTest.cpp：モックを用いた C++のテストコード

```
#include "CppUTest/TestHarness.h"
#include "CppUTestExt/MockSupport.h"

#include "gpio.h"
#include "key.h"

TEST_GROUP(key){
  TEST_SETUP(){
  }

  TEST_TEARDOWN(){
    mock().clear();
  }
};

TEST(key, Test_Mock)
{
  uint8_t key;

  mock()
      .expectOneCall("GPIO_ReadPin")
```

```
      .withUnsignedIntParameters("port", port)
      .withUnsignedIntParameters("pin", pin)
      .andReturnIntValue(0);
  key = Check_Key();
  LONGS_EQUAL(key, 1);

  // 結果検証
  mock().checkExpectations();
}

/** ---------------------------------------------------------------------------
 * @brief モック関数
 */
uint8_t GPIO_ReadPin(uint16_t port, uint8_t pin)
{
  mock()
      .actualCall("GPIO_ReadPin")
      .withUnsignedIntParameter("port", port)
      .withUnsignedIntParameter("pin", pin);
  return mock().returnIntValue();
}
```

7.3. 引数

引数の宣言は、型に合わせて次のものがあります。一部を除いて C と C++で記述方法が変わるので注意してください。

表 3　引数リスト

型	テストコード	モック関数
bool	withBoolParameters(const SimpleString& name, bool value) [C] withBoolParameter(const SimpleString& name, bool value) [C++]	
int	withIntParameters(const SimpleString& name, int value) [C] withIntParameter(const SimpleString& name, int value) [C++]	
unsigned int	withUnsignedIntParameters(const SimpleString& name, unsigned int value) [C] withUnsignedIntParameter(const SimpleString& name, unsigned int value) [C++]	
long	withLongIntParameters(const SimpleString& name, long value) [C] withLongIntParameter(const SimpleString& name, long value) [C++]	
unsigned long	withUnsignedLongIntParameters(const SimpleString& name, unsigned long value) [C] withUnsignedLongIntParameter(const SimpleString& name, unsigned long value) [C++]	
long long	withLongLongIntParameters(const SimpleString&, cpputest_longlong value) [C] withLongLongIntParameter(const SimpleString&, cpputest_longlong value) [C++]	
unsigned long long	withUnsignedLongLongIntParameters(const SimpleString& name, cpputest_ulonglong value) [C] withUnsignedLongLongIntParameter(const SimpleString& name, cpputest_ulonglong value) [C++]	
double	withDoubleParameters(const SimpleString& name, double value) [C] withDoubleParameter(const SimpleString& name, double value) [C++]	
char	withStringParameters(const SimpleString& name, const char* value) [C] withStringParameter(const SimpleString& name, const char* value) [C++]	
ポインター	withPointerParameters(const SimpleString& name, void* value) [C] withPointerParameter(const SimpleString& name, void* value) [C++]	
ポインター	withConstPointerParameters(const SimpleString& name, const void* value) [C] withConstPointerParameter(const SimpleString& name, const void* value) [C++]	
(値を返す)ポインター	withOutputParameter(const SimpleString& name, void* output) [C/C++]	
関数ポインター	withFunctionPointerParameters(const SimpleString& name, void (*value)()) [C] withFunctionPointerParameter(const SimpleString& name, void (*value)()) [C++]	

7.3.1. 独自の型

引数の型には表 3 以外に独自に定義した型を使用できます。この場合の記述方法が少し複雑なので、下に例を示します。なお下記例の記述方法が本当に正しいのかは、インターネットで検索しても例が殆どなく、正直よくわかっていません。

表 4　独自の型を引数にする場合

型	テストコード	モック関数
独自の型	withParameterOfType(const SimpleString& type, const SimpleString& name, const void* value) [C/C++]	
独自の型	withOutputParameterOfType(const SimpleString& type, const SimpleString& name, void* output) [C/C++]	

keyTest.cpp：モックを用いた C のテストコード

```
#include "CppUTest/TestHarness.h"
#include "CppUTestExt/MockSupport_c.h"

#include "gpio.h"
#include "key.h"

static int         equalMethod(const void *object1, const void *object2);  // @2
static const char *toStringMethod(const void *);  // @2

// clang-format off
TEST_GROUP(key){
  TEST_SETUP(){
  }

  TEST_TEARDOWN(){
    mock_c()->clear();  // モック資源のクリアー
  }
};
// clang-format on

TEST(key, Test_Mock)
{
  uint8_t key;

  mock_c()->installComparator("GPIO_TypeDef", equalMethod, toStringMethod);  // @1
  mock_c()
      ->expectOneCall("GPIO_ReadPin")
      ->withParameterOfType("GPIO_TypeDef", "port", &GPIOA)
      ->withUnsignedIntParameters("pin", KEY1)
      ->andReturnIntValue(0);

  key = Check_Key();
  LONGS_EQUAL(key, 1);

  mock_c()->removeAllComparatorsAndCopiers();  // @3

  // 結果検証
  mock_c()->checkExpectations();
}

/** ---------------------------------------------------------------------------
```

```
 * @brief モック関数
 */
uint8_t GPIO_ReadPin(GPIO_TypeDef *port, uint8_t pin)
{
  mock_c()
      ->actualCall("GPIO_ReadPin")
      ->withParameterOfType("GPIO_TypeDef", "port", port)
      ->withUnsignedIntParameters("pin", pin);
  return mock_c()->intReturnValue();
}

// @2
static int equalMethod(const void *object1, const void *object2)
{
  return object1 == object2;
}
// @2
static const char *toStringMethod(const void *)
{
  return "string";
}
```

表 5　解説

番号	解説
@1	独自の型を用いる場合、installComparator にて型を比較する方法などを渡す必要があります。
@2	installComparator で CppUMock に渡す関数です。何が正解かわかりませんが、このとおり記述しておけばとりあえず動きます。
@3	Comparator の情報を削除します。最後にこれがないと正常に動きません。

- CppUMock は C でしか使用したことがなく、C++での記述例はありません。

7.4. 戻り値

戻り値の宣言は、型に合わせて次のものがあります。C と C++でモック関数内の記述方法が変わるので注意してください。

表 6　戻り値リスト

型	テストコード	モック関数
bool	andReturnBoolValue	boolReturnValue [C] returnBoolValue [C++]
int	andReturnIntValue	intReturnValue [C] returnIntValue [C++]
unsigned int	andReturnUnsignedIntValue	unsignedIntReturnValue [C] returnUnsignedIntValue [C++]
long	andReturnLongIntValue	longIntReturnValue [C] returnLongIntValue [C++]
unsigned long	andReturnUnsignedLongIntValue	unsignedLongIntReturnValue [C] returnUnsignedLongIntValue [C++]
long long	andReturnLongLongIntValue	longLongIntReturnValue [C] returnLongLongIntValue [C++]
unsigned long long	andReturnUnsignedLongLongIntValue	nsignedLongLongIntReturnValue [C] returnUnsignedLongLongIntValue [C++]
double	andReturnDoubleValue	doubleReturnValue [C] returnDoubleValue [C++]
char	andReturnStringValue	stringReturnValue [C] returnStringValue [C++]
ポインター	andReturnPointerValue	pointerReturnValue [C] returnPointerValue [C++]
ポインター	andReturnConstPointerValue	constPointerReturnValue [C] returnConstPointerValue [C++]
関数ポインター	andReturnFunctionPointerValue	functionPointerReturnValue [C] returnFunctionPointerValue [C++]