SimpleCTester Framework Explained

Kjell Carlsson 2020-10-08

This is a unit test framework based on macros, for details check up SimpleCTester.h

User instructions

* Create separate folders for source code and test code in your project
* Create your test target code, as .h and .c files in your src folder
* Add file SimpleCTester.h to folder test
* Add template TestMain.c to folder test
* Define your tests following the template described in TestMain.c

Available framework tests

SimpleTest_AssertInteger(IntA, IntB, Message)
Test will check up that supplied integers IntA and IntB have the same value
If not, Message will be printed to output

SimpleTest_AssertString(StringA, StringB, Message)
Test will check up that supplied Strings have the same contents
If not, Message will be printed to output

SimpleTest_AssertTrue(Condition, Message)
Test will check up that supplied boolean Condition is true
If not, Message will be printed to output

SimpleTest_AssertFalse(Condition, Message)
Test will check up that supplied boolean Condition is false
If not, Message will be printed to output

All tests should be uniquely named and registered in function test_batch()

Running your version of TestMain.c will run all tests and provide a summary when done.

TestMainTemplate.c

```
/*
 *  TestMainTemplate.c
 *  Created on: 8 October 2020
 *  Author: Kjell H Carlsson
 */
#include <stdio.h>
#include "SimpleCTester.h"

/// supply path to .h file holding function headers of your test target
#include "<< PATH GOES HERE >>"

SimpleTest_Initialize()    // Counters for all tests are reset here

/****************************** Tests ******************************/
/// supply name of your test as parameter
SimpleTest_CreateTest(<< TEST NAME GOES HERE >>) // Single test defined
{
    /// define your test by calling functions from test target
    << TEST CODE GOES HERE >>

    /// assert that the outcome of your call matches expected outcome
    SimpleTest_AssertInteger(<<ACTUAL>>, <<EXPECTED>>, "<<ERROR MSG>>");
    /// or
    SimpleTest_AssertString(<<ACTUAL>>, <<EXPECTED>>, "<<ERROR MSG>>");
    /// or
    SimpleTest_AssertTrue(<<CONDITION>>, "<<ERROR MSG>>");
    SimpleTest_AssertFalse(<<CONDITION>>, "<<ERROR MSG>>");
}
SimpleTest_FinalizeTest()                        // End of test

/// Repeat the pattern above for all your tests for test target

/****************************** Batch ******************************/

// This is where we add up all tests to be run
static char* test_batch()
{
    /// supply info for presentation of outcome of test batch
    printf("\n Running tests for %s \n", "<< TARGET NAME >>");

    /// supply name of each test as parameter
    SimpleTest_RunTest(<< TEST NAME >>, "<< TEST NAME >>" ); // Run test
    SimpleTest_RunTest(<< TEST NAME >>, "<< TEST NAME >>");  // Run test
...
    return NULL;
}
```

```
********************** Test application main *************************/
int main(int argc, char **argv)
{
    test_batch();               // complete batch of tests will be run

    printf("\n Tests completed: %d", SimpleTestCounter); // tests run

    // if batch is successful
    if( SimpleTestFailCounter == 0){
        printf("\n All tests passed!");
    }
    // number of failed tests
    else{
       printf("\n Tests failed: %d", SimpleTestFailCounter);
    }

    return 0;
}
```

Example PrimeCheck

PrimeCheck.h

```
#ifndef PRIMECHECK_H_
#define PRIMECHECK_H_

#include <stdint.h>

uint32_t input_func();
uint32_t handle_even_factors(const uint32_t start_value);
void handle_odd_factors(uint32_t temp_value, uint32_t factor);
const char* create_output_msg();

#endif /* PRIMECHECK_H_ */
```

TestMain.c

```
#include <stdio.h>
#include "SimpleCTester.h"

/// supply path to .h file holding function headers of your test target
#include "..\\src\PrimeCheck.h"

extern uint16_t prime_count;

SimpleTest_Initialize()                // Counters for all tests are reset here
/********************************** Tests **********************************/

SimpleTest_CreateTest(Prime_Test1)     // Single test defined
{
    /// define your test by calling functions from test target
    uint32_t start_value = 11321;
    uint32_t temp_value = handle_even_factors(start_value);
    handle_odd_factors(temp_value, 1);

    /// assert that the outcome of your call matches expected outcome
    SimpleTest_AssertInteger(prime_count, 1, "Test value IS a Prime!");

}
SimpleTest_FinalizeTest()              // End of test

SimpleTest_CreateTest(No_Prime_Test1)  // Single test defined
{
    /// define your test by calling functions from test target
    uint32_t start_value = 32648;
    uint32_t temp_value = handle_even_factors(start_value);
    handle_odd_factors(temp_value, 1);
    const char* msg = create_output_msg();

    /// assert that the outcome of your call matches expected outcome
    SimpleTest_AssertString(msg, "No prime number", "Tested value is NOT a prime!");

}
SimpleTest_FinalizeTest()              // End of test
```

```
/// supply name of your test as parameter
SimpleTest_CreateTest(Prime_Test2)     // Single test defined - should fail!
{
    /// define your test by calling functions from test target
       uint32_t start_value = 31;
       uint32_t temp_value = handle_even_factors(start_value);
       handle_odd_factors(temp_value, 1);

    /// assert that the outcome of your call matches expected outcome
       SimpleTest_AssertFalse((prime_count != 1), "Test number IS a Prime!");

}
SimpleTest_FinalizeTest()

SimpleTest_CreateTest(No_Prime_Test2)  // Single test defined
{
    /// define your test by calling functions from test target
    uint32_t start_value = 821238;
    uint32_t temp_value = handle_even_factors(start_value);
    handle_odd_factors(temp_value, 1);

    /// assert that the outcome of your call matches expected outcome
    SimpleTest_AssertTrue((prime_count != 1), "Test number is NOT a Prime!");
}
SimpleTest_FinalizeTest()              // End of test

/********************************** Batch **********************************/

// This is where we add up all tests to be run
void test_batch()
{
    /// supply info for presentation of outcome of test batch
    printf("\n Running tests for %s \n", "PrimeCheck");

    /// supply name of each test as parameter
    SimpleTest_RunTest(Prime_Test1, "Prime_Test1");        // Run test
    SimpleTest_RunTest(No_Prime_Test1, "No_Prime_Test1");  // Run test
    SimpleTest_RunTest(Prime_Test2, "Prime_Test2");        // Run test
    SimpleTest_RunTest(No_Prime_Test2, "No_Prime_Test2");  // Run test

}

/************************ Test application main ***************************/
int main(int argc, char **argv)
{
    test_batch();                              // complete batch of tests will be run

    printf("\n Tests completed: %d", SimpleTestCounter);    // total number of tests run

    if( SimpleTestFailCounter == 0){
       printf("\n All tests passed!");                      // if batch is successful
    }
    else{
       printf("\n Tests failed: %d", SimpleTestFailCounter); // number of failed tests
    }
    return 0;
}
```

Output:

```
Running tests for PrimeCheck

11321
* Test no 1: Prime_Test1 passed

2 2 2 7 11 53
* Test no 2: No_Prime_Test1 passed

31
* Test no 3: Prime_Test2 FAILED Test number IS a Prime!

2 3 11 23 541
* Test no 4: No_Prime_Test2 passed

Tests completed: 4
Tests failed: 1
```

SimpleCTester.h

```
/*
 * SimpleCTester.h
 * Created on: 29 Sep 2019
 * Author: Kjell Carlsson
 */

#ifndef SRC_SIMPLECTESTER_H_
#define SRC_SIMPLECTESTER_H_

/******************************** Includes *********************************/
#include <string.h>
#include <stdio.h>

/************************** Constants and Types ****************************/
#define TRUE 1
#define FALSE 0

/********************************* Variables *******************************/
extern int SimpleTestCounter;
extern int SimpleTestFailCounter;

/************************** Exported Macro Functions ***********************/
#define SimpleTest_AssertInteger(IntA, IntB, Message) {
if(IntA != IntB) { ++SimpleTestFailCounter; return Message; } }

#define SimpleTest_AssertString(StringA, StringB, Message) {
if (strcmp(StringA, StringB) != 0) { ++SimpleTestFailCounter; return Message; } }

#define SimpleTest_AssertTrue(Condition, Message) {
if(Condition != TRUE) { ++SimpleTestFailCounter; return Message; } }

#define SimpleTest_AssertFalse(Condition, Message) {
if(Condition != FALSE) { ++SimpleTestFailCounter; return Message; } }


#define SimpleTest_Initialize() int SimpleTestCounter = 0u; int SimpleTestFailCounter = 0u;

#define SimpleTest_CreateTest(Name) static char* Name(void) {

#define SimpleTest_FinalizeTest() return NULL; }

#define SimpleTest_RunTest(test, Name) do{ char *message = test(); printf("\n * Test no
%d: %s ", ++SimpleTestCounter, Name); if(message) { printf("FAILED %s\n\n", message); }
else{ printf("passed\n\n"); } }while(FALSE);

#endif /* SRC_SIMPLECTESTER_H_ */
```