**CUNIT**

**A unit testing framework**

**CONTENTS**

1. **Unit Testing**
2. **CUnit**
   1. **Installing CUnit**
   2. **Structure**
   3. **Sequence to use CUnit framework in a project**
   4. **Example**
3. **UNIT TESTING**

* Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed.
* A unit is the smallest testable part of software. It usually has one or a few inputs and usually a single output. In procedural programming a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.)
* Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

1. **CUNIT**

* CUnit is a lightweight system for writing, administering, and running unit tests in C. It provides C programmers a basic testing functionality with a flexible variety of user interfaces.
* CUnit is built as a static library which is linked with the user's testing code. It uses a simple framework for building test structures, and provides a rich set of assertions for testing common data types. In addition, several different interfaces are provided for running tests and reporting results. These interfaces currently include :

|  |  |  |
| --- | --- | --- |
| 1. Automated | Output to xml file | Non-interactive |
| 1. Basic | Flexible programming interface | Non-interactive |
| 1. Console | Console interface (ANSI C) | Interactive |
| 1. Curses | Graphical interface (Unix) | Interactive |

* 1. **INSTALLING CUNIT**

1. Download the CUnit-2.1-2-src.tar.bz2 source from the <https://sourceforge.net/projects/cunit/> and save to temporary directory such as /tmp for example.
2. In a terminal window, unpack the CUnit-2.1-2-src.tar.bz2 file as follows:

$ cd /tmp

$ bunzip2 CUnit-2.1-2-src.tar.bz2

$ tar -xvf CUnit-2.1-2-src.tar

1. Configure, build, and install CUinit from /tmp as follows:

$ cd CUnit-2.1-2

$ ./configure --prefix=/usr

$ make

$ sudo make install

* 1. **STRUCTURE**
* Unit is a combination of a platform-independent framework with various user interfaces. The core framework provides basic support for managing a test registry, suites, and test cases. The user interfaces facilitate interaction with the framework to run tests and view results.
* CUnit is organized like a conventional unit testing framework:

Test Registry

|

------------------------------

| |

Suite '1' . . . . Suite 'N'

| |

--------------- ---------------

| | | |

Test '11' ... Test '1M' Test 'N1' ... Test 'NM'

* The core framework provides basic support for managing a test registry, suites, and test cases.
* The user interfaces facilitate interaction with the framework to run tests and view results.
* The core framework provides basic support for managing a test registry, suites, and test cases.
* The user interfaces facilitate interaction with the framework to run tests and view results.
  1. **SEQUENCE TO USE CUNIT FRAMEWORK IN A PROJECT**

1. Write functions for tests (and suite init/cleanup if necessary).
2. Initialize the test registry - CU\_initialize\_registry()

3. Add suites to the test registry - CU\_add\_suite()

4. Add tests to the suites - CU\_add\_test()

5. Run tests using an appropriate interface, e.g. CU\_console\_run\_tests

1. Cleanup the test registry - CU\_cleanup\_registry
   1. **EXAMPLE**

**max.c**

#include "max.h"

int max (int n1, int n2 )

{

if ( n2 > n1 ) return n2;

return n1;

}

**max.h**

#ifndef MAX\_H

#define MAX\_H

extern int max (int, int);

#endif

**max\_test.c**

#include "CUnit/CUnit.h"

#include "CUnit/Basic.h"

//#include "CUnit/Automated.h"

//#include "CUnit/Console.h"

#include "max.h"

#include <stdio.h> // for printf

/\* Test Suite setup and cleanup functions: \*/

int init\_suite(void) { return 0; }

int clean\_suite(void) { return 0; }

/\*\*\*\*\*\*\*\*\*\*\*\*\* Test case functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void test\_case\_sample(void)

{

CU\_ASSERT(CU\_TRUE);

CU\_ASSERT\_NOT\_EQUAL(2, -1);

CU\_ASSERT\_STRING\_EQUAL("string #1", "string #1");

CU\_ASSERT\_STRING\_NOT\_EQUAL("string #1", "string #2");

CU\_ASSERT(CU\_FALSE);

CU\_ASSERT\_EQUAL(2, 3);

CU\_ASSERT\_STRING\_NOT\_EQUAL("string #1", "string #1");

CU\_ASSERT\_STRING\_EQUAL("string #1", "string #2");

}

void max\_test\_1(void) {

CU\_ASSERT\_EQUAL( max(1,2), 2);

CU\_ASSERT\_EQUAL( max(2,1), 2);

}

void max\_test\_2(void) {

CU\_ASSERT\_EQUAL( max(2,2), 2);

CU\_ASSERT\_EQUAL( max(0,0), 0);

CU\_ASSERT\_EQUAL( max(-1,-1), -1);

}

void max\_test\_3(void) {

CU\_ASSERT\_EQUAL( max(-1,-2), -1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\* Test Runner Code goes here \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main ( void )

{

CU\_pSuite pSuite = NULL;

/\* initialize the CUnit test registry \*/

if ( CUE\_SUCCESS != CU\_initialize\_registry() )

return CU\_get\_error();

/\* add a suite to the registry \*/

pSuite = CU\_add\_suite( "max\_test\_suite", init\_suite, clean\_suite );

if ( NULL == pSuite ) {

CU\_cleanup\_registry();

return CU\_get\_error();

}

/\* add the tests to the suite \*/

if ( (NULL == CU\_add\_test(pSuite, "max\_test\_1", max\_test\_1)) ||

(NULL == CU\_add\_test(pSuite, "max\_test\_2", max\_test\_2)) ||

(NULL == CU\_add\_test(pSuite, "max\_test\_3", max\_test\_3))

)

{

CU\_cleanup\_registry();

return CU\_get\_error();

}

// Run all tests using the basic interface

CU\_basic\_set\_mode(CU\_BRM\_VERBOSE);

CU\_basic\_run\_tests();

printf("\n");

CU\_basic\_show\_failures(CU\_get\_failure\_list());

printf("\n\n");

/\*

// Run all tests using the automated interface

CU\_automated\_run\_tests();

CU\_list\_tests\_to\_file();

// Run all tests using the console interface

CU\_console\_run\_tests();

\*/

/\* Clean up registry and return \*/

CU\_cleanup\_registry();

return CU\_get\_error();