libroman

1.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

libroman - Documentation

This project was made as an assignment for the course *Métodos de Programação 1* (Programming Methods 101), Computing Department, UnB, April 22nd, 2017. It consists in a library to convert Roman numbers to their integer values.

See the README for details about how to build, run and test the library and a little explanation about the inner workings of the Roman numbers.

Development

I used the test-driven development (TDD) for the coding of this library. The tests were coded (and then the needed code to satisfy them) and developed as in the sucession below:

1. testSimple, where I check if basic input strings are correctly handled.
2. testIncreasing, where I check if the case in which we have only increasing algarisms.
3. testSubtracting, where I check the case for subtracting algarisms, i.e., IV or XM.
4. testUntil3000, where I check from a list from 1 to 3000.
5. testBin, where I check the standalone binary for inconsistencies.

The final version of the code library can be found in src/roman.c and in src/roman-to-int.c , the final version of the standalone binary.

Test code coverage

I then checked the test coverage using the gcov tool, yielding the following results:

```
gcov obj/roman.o
File 'src/roman.c'
Lines executed:100.00% of 64

gcov obj/roman-to-int.o
File 'src/roman-to-int.c'
Lines executed:90.48% of 21
```

In the src/roman-to-int.c file, the non-executed lines are referring to the check in case malloc() fails, which is a very unlikely case.

```
if(arg_str == NULL) {
    fprintf(stderr, "Couldn't allocate memory.\n");
    return 1;
}
```

The corresponding .gcov files can be found in the following links: roman.c.gcov and roman-to-int.c.gcov

Static analysis

I ran the command `cppcheck --enable=warning .` in the source code directory, obtaining the following results:

```
Checking src/roman-to-int.c...
1/7 files checked 14% done
Checking src/roman.c...
2/7 files checked 28% done
Checking test/testBin.c...
3/7 files checked 42% done
Checking test/testIncreasing.c...
4/7 files checked 57% done
Checking test/testSimple.c...
5/7 files checked 71% done
Checking test/testSubtracting.c...
6/7 files checked 85% done
Checking test/testUntil3000.c...
7/7 files checked 100% done
```

Chapter 2

libroman

A library to convert Roman numerals to their integer values.

Building the library

To build the static library, you'll need the following commands available in your system:

- `g++` (C++ compiler)
- `ar` (static library creator)
- `pthreads`
- `cmake`

In addition, to run the tests you need to have the GTest framework in your system. The easiest way is to clone the `Google Test repository` to a directory in your local path and then build it using `cmake`:

```
1 cd /home/foo/
2 git clone https://github.com/google/googletest
3 cd googletest
4 mkdir build && cd build
5 cmake .. && make
```

You can then `cd` to this project root folder `libroman` and pass the GTest root directory as the variable `GTEST_ROOT_DIR` to the `make` command.

```
1 cd /home/foo/libroman
2 make GTEST_ROOT_DIR=/home/foo/googletest
```

If not, you'll need to provide the paths to GTest as variables to the `make` command:

```
1 cd /home/foo/libroman
2 make GTEST_LIB_DIR=/home/foo/gtest-lib-path GTEST_INCLUDE_DIR=/home/foo/gtest-include-path
```

- `GTEST_LIB_DIR` points to where the library .a file is. Defaults to `GTEST_ROOT_DIR/build/googlemock/gtest`.
- `GTEST_INCLUDE_DIR` must point to where the include directory `gtest` is. `Defaults to GTEST_ROOT_DIR/googletest/include`.

To run the tests, simply run `make run-tests`.

Using the library

To use the library, simply include the header roman.h and call the function roman_to_int() with a string as its only parameter:

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

int main() {
    int value = roman_to_int("XII");
    printf("%d\n", value); // 12
}
```

And then you need to compile your program linking it against the `libroman.a` static library in the `lib/` directory, including the header directory as well:

```
1 g++ -o bar bar.c -I /home/foo/libroman/include -L /home/foo/libroman/lib -lroman
```

Using the standalone binary

There's a standalone binary `roman-to-int` in the `bin/` directory that accepts Roman numerical strings as its only command-line argument, and prints to stdout its numerical integer value. For example, the code below outputs `12`:

```
1 cd /home/foo/libroman
2 ./bin/roman-to-int XII
```

Documentation

To read the docs, go to `doc/html/index.html` in your browser or open the file `doc/latex/refman.pdf`.

Roman numerical system

The Roman numerical system is composed by assigning numerical values to certain characters, as in the table below:

Character	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

The numbers must normally be written in descending order, with repeating characters having their value multiplied by the number of times they appear (3, at most). As such, we'd have VI = 6, MMMLX = 3060, for instance. The characters V, L and D cannot be repeated in any circunstance, rendering Roman numbers such as VV or MDDI invalid.

If a character like I, X or C appears before another with a greater value, its value is subtracted from this one: IV = 4, for instance. The rules for subtracting characters are written in the table below:

Character	Can appear before
I	V, X
X	L, C
C	D, M
V, L, D	None

When subtracting there can't be any repetition, thus IIX or CCMI are invalid Roman numbers.

Chapter 3

Simple tests

Starting with tests for simple inputs: test/testSimple.c

First test - sanity of the input

The first test I did just checks if the function handles correctly non-valid strings i.e., empty or NULL strings:

```
TEST(SimpleInput, InvalidInput) {
    EXPECT_EQ(roman_to_int(NULL), -1);
    EXPECT_EQ(roman_to_int(""), -1);
}
```

Second test - single characters

The second test involves checking if the conversion for single characters is working as expected:

```
TEST(SimpleInput, SingleCharacter) {
    EXPECT_EQ(roman_to_int("I"), 1);
    EXPECT_EQ(roman_to_int("V"), 5);
    EXPECT_EQ(roman_to_int("X"), 10);
    EXPECT_EQ(roman_to_int("L"), 50);
    EXPECT_EQ(roman_to_int("C"), 100);
    EXPECT_EQ(roman_to_int("D"), 500);
    EXPECT_EQ(roman_to_int("M"), 1000);
}
```

The resulting code after these tests can be seen in this `GitHub commit`. In a nutshell, I added a test for the validity of the string, making the function `roman_to_int()` pass the first test. Then I added the function `single_character_value()` to return the numerical value of a Roman character, satisfying the second test.

Chapter 4

Increasing characters

Tests for more complex inputs: test/testIncreasing.c

First test - multiple single characters

The first test I did checks if the function is returning the right values for strings with repeatable characters (I, X, C and M).

```
TEST(ComplexInput, MultipleSingleCharacters) {
    EXPECT_EQ(roman_to_int("II"), 2);
    EXPECT_EQ(roman_to_int("XXX"), 30);
    EXPECT_EQ(roman_to_int("L"), 50);
    EXPECT_EQ(roman_to_int("CC"), 200);
    EXPECT_EQ(roman_to_int("CCC"), 300);
    EXPECT_EQ(roman_to_int("MM"), 2000);
}
```

This test was satisfied by creating a loop through the string to add all the values in it, as can be seen in this GitHub commit.

Second test - increasing characters

The first test I did just checks if the function handles correctly inputs with multiple, but strictly increasing values for its algarisms.

```
TEST(ComplexInput, IncreasingCharacters) {
    EXPECT_EQ(roman_to_int("XI"), 11);
    EXPECT_EQ(roman_to_int("XII"), 12);
    EXPECT_EQ(roman_to_int("VI"), 6);
    EXPECT_EQ(roman_to_int("LX"), 60);
    EXPECT_EQ(roman_to_int("MDXX"), 1520);
}
```

Third test - non-repeatable characters

This snippet checks if the function fails (as expected) for strings where one of the non-repeatable characters (V, L or D) appear more than once.

```
TEST(ComplexInput, NonRepeatableCharacters) {
    EXPECT_EQ(roman_to_int("VV"), -1);
    EXPECT_EQ(roman_to_int("DDD"), -1);
    EXPECT_EQ(roman_to_int("LL"), -1);
    EXPECT_EQ(roman_to_int("MDDX"), -1);
}
```

Fourth test - too many repetitions

This snippet checks if the function fails when the repeatable characters (I, X, C or D) repeat more than 3 times.

```
TEST(ComplexInput, TooManyRepetitions) {
    EXPECT_EQ(roman_to_int("XXXXX"), -1);
    EXPECT_EQ(roman_to_int("MMMM"), -1);
    EXPECT_EQ(roman_to_int("IIIIIIII"), -1);

    EXPECT_EQ(roman_to_int("MIIII"), -1);
    EXPECT_EQ(roman_to_int("DXXXX"), -1);
    EXPECT_EQ(roman_to_int("LXXXXX"), -1);
    EXPECT_EQ(roman_to_int("MMMMMDXII"), -1);
}
```

To satisfy the 2nd, 3rd and 4th tests, the function was altered as in this Github commit. I added a check function to see if the character is repeatable and then, in the main `roman_to_int()` function, I checked if the repeatable ones weren't repeating too much (more than 3 times).

Fifth test - invalid characters

This test checks if the function behaves correctly when a string with invalid characters is passed as the input:

```
TEST(ComplexInput, InvalidCharacter) {
    EXPECT_EQ(roman_to_int("A"), -1);
    EXPECT_EQ(roman_to_int("XXe"), -1);
    EXPECT_EQ(roman_to_int("xx"), -1);
    EXPECT_EQ(roman_to_int("AMMMI"), -1);
    EXPECT_EQ(roman_to_int("C "), -1);
    EXPECT_EQ(roman_to_int("D  I"), -1);
    EXPECT_EQ(roman_to_int("MM_1"), -1);
}
```

In this Github commit, I made the function comply to this test by making it return `-1` when encountering an invalid character.

Chapter 5

Subtracting

Tests for subtracting characters: test/testSubtracting.c

First test - wrong order

This test checks if the function is failing for subtracting characters in the wrong order, i.e.: V, L or D before a higher value algarism.

```
TEST(SubtractingInput, WrongOrder) {
    EXPECT_EQ(roman_to_int("ICCV"), -1);
    EXPECT_EQ(roman_to_int("IM"), -1);
    EXPECT_EQ(roman_to_int("IDX"), -1);

    EXPECT_EQ(roman_to_int("XVX"), -1);
    EXPECT_EQ(roman_to_int("DM"), -1);

    EXPECT_EQ(roman_to_int("LM"), -1);
    EXPECT_EQ(roman_to_int("LCIX"), -1);
}
```

Second test - negative characters

This one confirms if the function is working correctly for characters with negative value, i.e.: I appearing V or X, C appearing before D or M, etc.

```
TEST(SubtractingInput, SingleSubtractingChar) {
    EXPECT_EQ(roman_to_int("IV"), 4);
    EXPECT_EQ(roman_to_int("IX"), 9);
    EXPECT_EQ(roman_to_int("XCII"), 92);
    EXPECT_EQ(roman_to_int("CMLXX"), 970);
}
```

Both tests were satisfied when I created a helper function to check if the algarism can come before another. Git↩ Hub commit

Third test - multiple negative characters

This snippet checks if the function fails (as expected) for strings where a negative-valued algarism appears more than once.

```
TEST(SubtractingInput, MultipleSubtractingChar) {
    EXPECT_EQ(roman_to_int("IIX"), -1);
    EXPECT_EQ(roman_to_int("IIIV"), -1);
    EXPECT_EQ(roman_to_int("XXCVI"), -1);
    EXPECT_EQ(roman_to_int("XXCIV"), -1);
}
```

I made the function satisfy this test by checking if there are repetitions when the function is already in the subtracting mode: GitHub commit

Chapter 6

Until 3000

Tests for a list until 3000: test/testUntil3000.c

First test - until 3000

This test reads a table with the Roman numerals from 1 to 3000, to see if the function is working for all these values:
GitHub commit

```
TEST(TestingList, TestingUntil3000) {
    FILE *fp = fopen("test/data/romans-until-3000.dat", "r");
    ASSERT_TRUE(fp != NULL);
    char buffer[LINE_MAX_SIZE];

    while(fgets(buffer, LINE_MAX_SIZE, fp)) {
        int value;
        char roman[LINE_MAX_SIZE];

        if(sscanf(buffer, "%d %49s ", &value, roman) != 2)
            continue;

        EXPECT_EQ(roman_to_int(roman), value);
    }

    fclose(fp);
}
```

Chapter 7

Standalone binary

Simple tests for the standalone binary

First test - standalone binary

This test just checks if the standalone binary fails when it's passed the wrong command-line arguments. GitHub commit

```
TEST(TestingBin, ExitsGracefully) {
    EXPECT_NE(system("./bin/roman-to-int"), 0);
    EXPECT_NE(system("./bin/roman-to-int VI 38 --system"), 0);

    EXPECT_EQ(system("./bin/roman-to-int VI"), 0);
    EXPECT_EQ(system("./bin/roman-to-int mIiI"), 0);

    EXPECT_NE(system("./bin/roman-to-int IM"), 0);
    EXPECT_NE(system("./bin/roman-to-int XDI"), 0);
}
```

Chapter 8

File Index

8.1 File List

Here is a list of all files with brief descriptions:

Chapter 9

File Documentation

9.1 doc/testBin.md File Reference

9.2 doc/testIncreasing.md File Reference

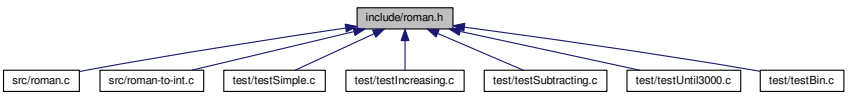
9.3 doc/testSimple.md File Reference

9.4 doc/testSubtracting.md File Reference

9.5 doc/testUntil3000.md File Reference

9.6 include/roman.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int roman_to_int (const char *input)

  Returns the integer value of the roman characters string.

9.6.1 Function Documentation

9.6.1.1 int roman_to_int (const char * *input*)

Returns the integer value of the roman characters string.

This function parses the input string, which must correspond to a valid roman character string, which must satisfy the following conditions:

- It must be non-null and non-empty.
- The characters V, L or D cannot be repeated.
- The characters I, X, C or M can be repeated up to three times.
- If the characters are in non-decreasing order

Definition at line 101 of file roman.c.

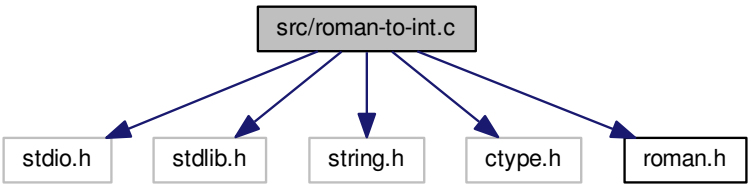
9.7 mainpage.dox File Reference

9.8 README.md File Reference

9.9 src/roman-to-int.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "roman.h"
```

Include dependency graph for roman-to-int.c:



Functions

- int main (int argc, char *argv[ ])

9.9.1 Function Documentation

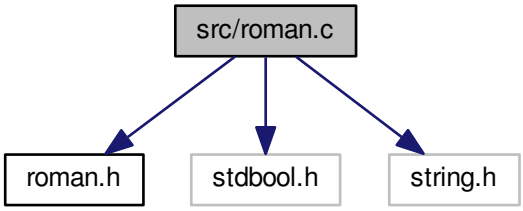
9.9.1.1 int main (int *argc*, char * *argv[]*)

Definition at line 8 of file roman-to-int.c.

9.10 src/roman.c File Reference

```
#include "roman.h"
#include <stdbool.h>
#include <string.h>
```

Include dependency graph for roman.c:



Functions

- static int string_length (const char *str)

  Returns the length of a valid string.
- static int alg_value (char alg)

  Returns the integer value of a single Roman algarism.
- static bool alg_can_repeat (char alg)

  Checks if the Roman algarism is repeatable (i.e., it's not 'V', 'L' or 'D').
- static bool alg_can_come_before (char now, char before)

  Checks if the Roman algarism `now` can come before the character `before` in a valid string, meaning its value is negative.
- int roman_to_int (const char *input)

  Returns the integer value of the roman characters string.

9.10.1 Function Documentation

9.10.1.1 static bool alg_can_come_before (char *now*, char *before*) [static]

Checks if the Roman algarism `now` can come before the character `before` in a valid string, meaning its value is negative.

The rules for such preceding characters are as below:

Character	Can appear before
'I'	V, X
'X'	L, C
'C'	D, M
'V', 'L', 'D'	None

Definition at line 85 of file roman.c.

9.10.1.2 static bool alg_can_repeat (char *alg*) [static]

Checks if the Roman algarism is repeatable (i.e., it's not 'V', 'L' or 'D').

Definition at line 69 of file roman.c.

9.10.1.3 static int alg_value (char *alg*) [static]

Returns the integer value of a single Roman algarism.

The corresponding value is given in the table below:

Character	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Definition at line 34 of file roman.c.

9.10.1.4 int roman_to_int (const char * *input*)

Returns the integer value of the roman characters string.

This function parses the input string, which must correspond to a valid roman character string, which must satisfy the following conditions:

- It must be non-null and non-empty.
- The characters V, L or D cannot be repeated.
- The characters I, X, C or M can be repeated up to three times.
- If the characters are in non-decreasing order

Definition at line 101 of file roman.c.

9.10.1.5 static int string_length (const char * *str*) [static]

Returns the length of a valid string.

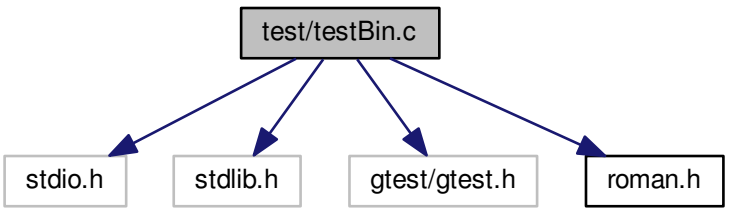
If the string is not valid (i.e., NULL or empty), returns -1.

Definition at line 12 of file roman.c.

9.11 test/testBin.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "gtest/gtest.h"
#include "roman.h"
```

Include dependency graph for testBin.c:



Functions

- TEST (TestingBin, ExitsGracefully)
  [TestingBin, ExitsGracefully]
- int main (int argc, char **argv)
  [TestingBin, ExitsGracefully]

9.11.1 Function Documentation

9.11.1.1 int main (int *argc,* char ** *argv*)

[TestingBin, ExitsGracefully]

Definition at line 20 of file testBin.c.

9.11.1.2 TEST (TestingBin , ExitsGracefully)

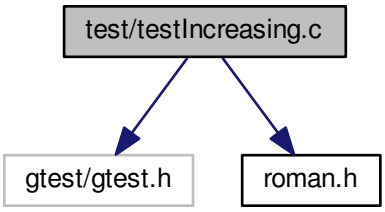
[TestingBin, ExitsGracefully]

Definition at line 8 of file testBin.c.

9.12 test/testIncreasing.c File Reference

```
#include "gtest/gtest.h"
#include "roman.h"
```

Include dependency graph for testIncreasing.c:



Functions

- TEST (ComplexInput, IncreasingCharacters)
  [ComplexInput, IncreasingCharacters]
- TEST (ComplexInput, InvalidCharacter)
  [ComplexInput, IncreasingCharacters]
- TEST (ComplexInput, MultipleSingleCharacters)
  [ComplexInput, InvalidCharacter]
- TEST (ComplexInput, NonRepeatableCharacters)
  [ComplexInput, MultipleSingleCharacters]
- TEST (ComplexInput, TooManyRepetitions)
  [ComplexInput, NonRepeatableCharacters]
- int main (int argc, char **argv)
  [ComplexInput, TooManyRepetitions]

9.12.1 Function Documentation

9.12.1.1 int main (int *argc*, char ** *argv*)

[ComplexInput, TooManyRepetitions]

Definition at line 59 of file testIncreasing.c.

9.12.1.2 TEST (ComplexInput , IncreasingCharacters)

[ComplexInput, IncreasingCharacters]

Definition at line 5 of file testIncreasing.c.

9.12.1.3 TEST (ComplexInput , InvalidCharacter)

[ComplexInput, IncreasingCharacters]

[ComplexInput, InvalidCharacter]

Definition at line 15 of file testIncreasing.c.

9.12.1.4 TEST (ComplexInput , MultipleSingleCharacters)

[ComplexInput, InvalidCharacter]

[ComplexInput, MultipleSingleCharacters]

Definition at line 27 of file testIncreasing.c.

9.12.1.5 TEST (ComplexInput , NonRepeatableCharacters)

[ComplexInput, MultipleSingleCharacters]

[ComplexInput, NonRepeatableCharacters]

Definition at line 38 of file testIncreasing.c.

9.12.1.6 TEST (ComplexInput , TooManyRepetitions)

[ComplexInput, NonRepeatableCharacters]

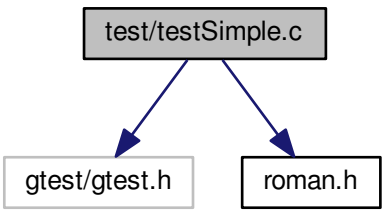
[ComplexInput, TooManyRepetitions]

Definition at line 47 of file testIncreasing.c.

9.13 test/testSimple.c File Reference

```
#include "gtest/gtest.h"
#include "roman.h"
```

Include dependency graph for testSimple.c:



Functions

- TEST (SimpleInput, InvalidInput)
  [SimpleInput, InvalidInput]
- TEST (SimpleInput, SingleCharacter)
  [SimpleInput, InvalidInput]
- int main (int argc, char **argv)
  [SimpleInput, SingleCharacter]

9.13.1 Function Documentation

9.13.1.1 int main (int *argc*, char ** *argv*)

[SimpleInput, SingleCharacter]

Definition at line 23 of file testSimple.c.

9.13.1.2 TEST (SimpleInput , InvalidInput)

[SimpleInput, InvalidInput]

Definition at line 5 of file testSimple.c.

9.13.1.3 TEST (SimpleInput , SingleCharacter)

[SimpleInput, InvalidInput]

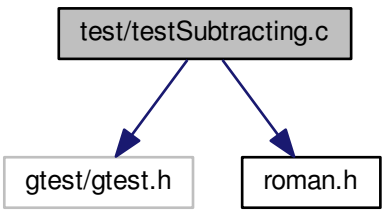
[SimpleInput, SingleCharacter]

Definition at line 12 of file testSimple.c.

9.14 test/testSubtracting.c File Reference

```
#include "gtest/gtest.h"
#include "roman.h"
```

Include dependency graph for testSubtracting.c:



Functions

- TEST (SubtractingInput, WrongOrder)

  [SubtractingInput, WrongOrder]
- TEST (SubtractingInput, SingleSubtractingChar)

  [SubtractingInput, WrongOrder]
- TEST (SubtractingInput, MultipleSubtractingChar)

  [SubtractingInput, SingleSubtractingChar]
- int main (int argc, char **argv)

  [SubtractingInput, MultipleSubtractingChar]

9.14.1 Function Documentation

9.14.1.1 int main (int *argc*, char ** *argv*)

[SubtractingInput, MultipleSubtractingChar]

Definition at line 36 of file testSubtracting.c.

9.14.1.2 TEST (SubtractingInput , WrongOrder)

[SubtractingInput, WrongOrder]

Definition at line 5 of file testSubtracting.c.

9.14.1.3 TEST (SubtractingInput , SingleSubtractingChar)

[SubtractingInput, WrongOrder]

[SubtractingInput, SingleSubtractingChar]

Definition at line 19 of file testSubtracting.c.

9.14.1.4 TEST (SubtractingInput , MultipleSubtractingChar)

[SubtractingInput, SingleSubtractingChar]

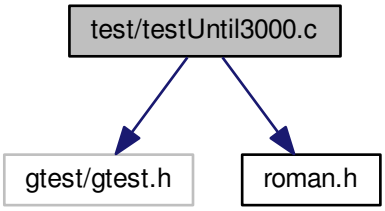
[SubtractingInput, MultipleSubtractingChar]

Definition at line 28 of file testSubtracting.c.

9.15 test/testUntil3000.c File Reference

```
#include "gtest/gtest.h"
#include "roman.h"
```

Include dependency graph for testUntil3000.c:



Macros

- #define LINE_MAX_SIZE 50

Functions

- TEST (TestingList, TestingUntil3000)
  [TestingList, TestingUntil3000]
- int main (int argc, char **argv)
  [TestingList, TestingUntil3000]

9.15.1 Macro Definition Documentation

9.15.1.1 #define LINE_MAX_SIZE 50

Definition at line 4 of file testUntil3000.c.

9.15.2 Function Documentation

9.15.2.1 int main (int *argc,* char ** *argv*)

[TestingList, TestingUntil3000]

Definition at line 26 of file testUntil3000.c.

9.15.2.2 TEST (TestingList , TestingUntil3000)

[TestingList, TestingUntil3000]

Definition at line 7 of file testUntil3000.c.

Index