cryptopals_c

Generated by Doxygen 1.9.8

Chapter 1

Cryptopals in C Code

Solve the **Cryptopals Crypto Challenges** using **pure C**.

- This repository contains my implementations, notes, and explanations for each challenge set, focused on clarity, correctness, and learning modern applied cryptography from first principles.

Basically a quiet Saturday and I am in need of a challenge, so I am speedrunning the Cryptopals challenges in C code, just for fun.

1.1 Overview

The **Cryptopals Challenges** are a well-known set of practical cryptography exercises covering topics like:

- Encoding and decoding
- XOR ciphers
- Block ciphers (AES)
- Padding oracles
- CBC, CTR, stream attacks
- Diffie–Hellman, RSA, and more

This repo is my attempt to solve them **from scratch in C**, without relying on high-level crypto libraries unless the challenge's intent allows it.
The goal is to deeply understand how the primitives work internally.

1.2 Requirements

- **C99 or newer**
- **Make**
- **doxygen and pdflatex/texlive environment for docs**

1.3 Building & Running

Common combinations of the below:

```
make clean
make
make test
make docs
```

1.4 Security Disclaimer

- This code is for my own amusement and educational purposes only.
- Do not use it in production, security-critical, or cryptographic applications.

1.5 References

Cryptopals Challenges

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

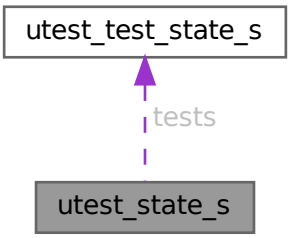
Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 utest_state_s Struct Reference

`#include <utest.h>`
Collaboration diagram for utest_state_s:



Data Fields

- struct utest_test_state_s * tests
- size_t tests_length
- FILE * output

4.1.1 Field Documentation

4.1.1.1 output

`FILE* utest_state_s::output`

4.1.1.2 tests

`struct utest_test_state_s* utest_state_s::tests`

4.1.1.3 tests_length

`size_t utest_state_s::tests_length`
The documentation for this struct was generated from the following file:

- utest.h

4.2 utest_test_state_s Struct Reference

`#include <utest.h>`

Data Fields

- utest_testcase_t func
- size_t index
- char * name

4.2.1 Field Documentation

4.2.1.1 func

`utest_testcase_t utest_test_state_s::func`

4.2.1.2 index

`size_t utest_test_state_s::index`

4.2.1.3 name

`char* utest_test_state_s::name`

The documentation for this struct was generated from the following file:

- utest.h

Chapter 5

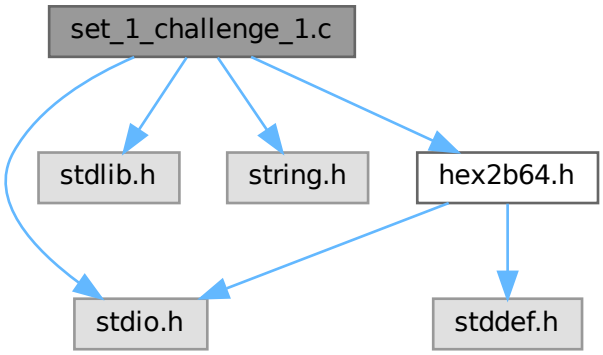
File Documentation

5.1 set_1_challenge_1.c File Reference

Cryptopals Set 1 Challenge 1 sample solution using hex2b64_buffer().

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hex2b64.h"
```

Include dependency graph for set_1_challenge_1.c:



Functions

- int main (void)

  Convert the fixed challenge hex string and compare to expected Base64.

5.1.1 Detailed Description

Cryptopals Set 1 Challenge 1 sample solution using hex2b64_buffer().

5.1.2 Function Documentation

5.1.2.1 main()

```
int main (
            void  )
```

Convert the fixed challenge hex string and compare to expected Base64.

Returns

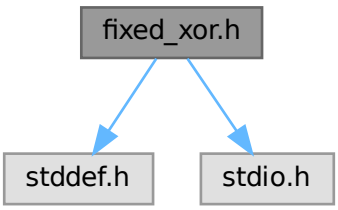
EXIT_SUCCESS when the conversion matches the known answer.

5.2 fixed_xor.h File Reference

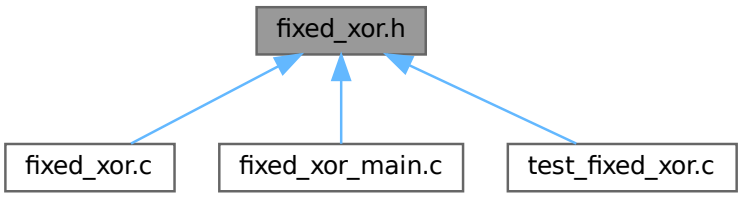
Public interface for XORing two equally sized buffers.

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

Include dependency graph for fixed_xor.h:



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



Enumerations

- enum fixed_xor_status {
  FIXED_XOR_OK = 0 , FIXED_XOR_ERR_ARGS = -1 , FIXED_XOR_ERR_IO = -2 , FIXED_XOR_ERR_ODD_INPUT = -3 ,
  FIXED_XOR_ERR_OOM = -4 }

  Status codes describing the outcome of fixed XOR operations.

Functions

- fixed_xor_status fixed_xor_buffers (const unsigned char *lhs, const unsigned char *rhs, unsigned char *out, size_t len)

  XOR two buffers of equal length into an output buffer.
- fixed_xor_status fixed_xor_stream (FILE *in, FILE *out)

XOR two half-length buffers read sequentially from a stream.

- const char * fixed_xor_status_string (fixed_xor_status status)

  Convert a fixed_xor_status value into a human-readable string.

5.2.1 Detailed Description

Public interface for XORing two equally sized buffers.

5.2.2 Enumeration Type Documentation

5.2.2.1 fixed_xor_status

```
enum fixed_xor_status
```

Status codes describing the outcome of fixed XOR operations.

Enumerator

FIXED_XOR_OK	Operation completed successfully.
FIXED_XOR_ERR_ARGS	Invalid arguments were supplied.
FIXED_XOR_ERR_IO	I/O failure while reading or writing.
FIXED_XOR_ERR_ODD_INPUT	Stream input contained an odd byte count.
FIXED_XOR_ERR_OOM	Memory allocation failed.

5.2.3 Function Documentation

5.2.3.1 fixed_xor_buffers()

```
fixed_xor_status fixed_xor_buffers (
            const unsigned char * lhs,
            const unsigned char * rhs,
            unsigned char * out,
            size_t len )
```

XOR two buffers of equal length into an output buffer.

Parameters

lhs	Pointer to the left-hand buffer.
rhs	Pointer to the right-hand buffer.
out	Destination buffer that receives lhs ^ rhs.
len	Number of bytes to process.

Returns

FIXED_XOR_OK on success or an error status on failure.

XOR two buffers of equal length into an output buffer.

5.2.3.2 fixed_xor_status_string()

```
const char * fixed_xor_status_string (
            fixed_xor_status status )
```

Convert a fixed_xor_status value into a human-readable string.

Parameters

status	Status code to describe.

Returns

Pointer to a static string literal.

Convert a fixed_xor_status value into a human-readable string.

5.2.3.3 fixed_xor_stream()

```
fixed_xor_status fixed_xor_stream (
            FILE * in,
            FILE * out )
```

XOR two half-length buffers read sequentially from a stream.
The input stream is expected to contain two equally sized buffers back to back. The result is written to `out`.

Parameters

in	Stream containing the concatenated buffers.
out	Stream that receives XOR output.

Returns

FIXED_XOR_OK on success or an error status on failure.

XOR two half-length buffers read sequentially from a stream.

5.3 fixed_xor.h

Go to the documentation of this file.

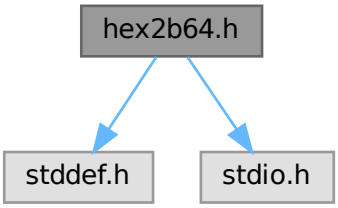
```
00001 #ifndef FIXED_XOR_H
00002 #define FIXED_XOR_H
00003
00009 #include <stddef.h>
00010 #include <stdio.h>
00011
00015 typedef enum {
00016     FIXED_XOR_OK = 0,
00017     FIXED_XOR_ERR_ARGS = -1,
00018     FIXED_XOR_ERR_IO = -2,
00019     FIXED_XOR_ERR_ODD_INPUT = -3,
00020     FIXED_XOR_ERR_OOM = -4
00021 } fixed_xor_status;
00022
00032 fixed_xor_status fixed_xor_buffers(const unsigned char *lhs,
00033                                    const unsigned char *rhs,
00034                                    unsigned char *out,
00035                                    size_t len);
00036
00047 fixed_xor_status fixed_xor_stream(FILE *in, FILE *out);
00048
00055 const char *fixed_xor_status_string(fixed_xor_status status);
00056
00057 #endif /* FIXED_XOR_H */
```

5.4 hex2b64.h File Reference

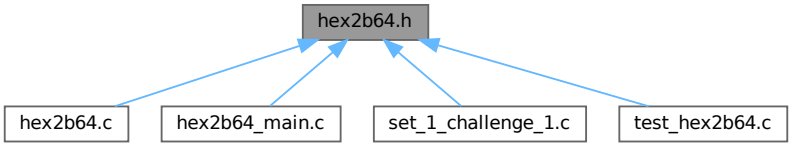
Public interface for converting hexadecimal data to Base64.

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

Include dependency graph for hex2b64.h:



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



Functions

- int hex2b64_stream (FILE ∗in, FILE ∗out)

  Convert hexadecimal text read from a stream into Base64.
- int hex2b64_buffer (const unsigned char ∗hex, size_t hex_len, unsigned char ∗out, size_t out_cap, size_t ∗out_len)

  Convert a memory buffer of hexadecimal characters into Base64.

5.4.1 Detailed Description

Public interface for converting hexadecimal data to Base64.

5.4.2 Function Documentation

5.4.2.1 hex2b64_buffer()

```
int hex2b64_buffer (
            const unsigned char * hex,
            size_t hex_len,
            unsigned char * out,
            size_t out_cap,
            size_t * out_len )
```

Convert a memory buffer of hexadecimal characters into Base64.
The input buffer may contain ASCII whitespace, which is skipped. The output buffer receives the Base64 encoding followed by a newline. The caller must supply sufficient capacity (including room for the newline). When `out_len` is non-NULL it receives the number of bytes written.

Parameters

hex	Pointer to the hex buffer (may be NULL when `hex_len` is 0).
hex_len	Number of bytes in `hex`.
out	Destination buffer for Base64 characters.
out_cap	Capacity of `out` in bytes.
out_len	Optional pointer that receives the bytes produced.

Returns

0 on success, non-zero on invalid hex, odd digit count, or overflow.

Convert a memory buffer of hexadecimal characters into Base64.

5.4.2.2 hex2b64_stream()

```
int hex2b64_stream (
            FILE * in,
            FILE * out )
```

Convert hexadecimal text read from a stream into Base64.
Whitespace characters in the input stream are ignored. The output stream receives the Base64 encoding plus a trailing newline. Errors are reported via the return code and (for malformed hex) an explanatory message on stderr.

Parameters

in	Input stream providing ASCII hex characters.
out	Output stream that receives Base64 data.

Returns

0 on success, non-zero on invalid input or I/O failure.

Convert hexadecimal text read from a stream into Base64.

5.5 hex2b64.h

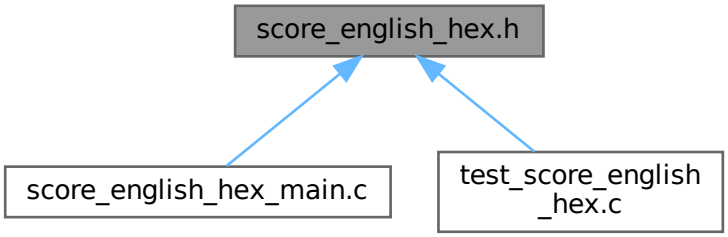
Go to the documentation of this file.

```
00001 #ifndef HEX2B64_H
00002 #define HEX2B64_H
00003
00009 #include <stddef.h>
00010 #include <stdio.h>
00011
00023 int hex2b64_stream(FILE *in, FILE *out);
00024
00040 int hex2b64_buffer(const unsigned char *hex,
00041                    size_t hex_len,
00042                    unsigned char *out,
00043                    size_t out_cap,
00044                    size_t *out_len);
00045
00046 #endif /* HEX2B64_H */
```

5.6 score_english_hex.h File Reference

Estimate how closely hex-encoded data resembles English text.

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



Functions

- double score_english_hex (const char *hex)

  Score a hex string based on English letter frequency heuristics.

5.6.1 Detailed Description

Estimate how closely hex-encoded data resembles English text.

5.6.2 Function Documentation

5.6.2.1 score_english_hex()

```
double score_english_hex (
            const char * hex )
```

Score a hex string based on English letter frequency heuristics.

Parameters

hex	Null-terminated ASCII hex string.

Returns

Higher values indicate closer resemblance to English text.

Score a hex string based on English letter frequency heuristics.

5.7 score_english_hex.h

Go to the documentation of this file.

```
00001 #ifndef SCORE_ENGLISH_HEX_H
00002 #define SCORE_ENGLISH_HEX_H
00003
00015 double score_english_hex(const char *hex);
00016
00017 #endif /* SCORE_ENGLISH_HEX_H */
```

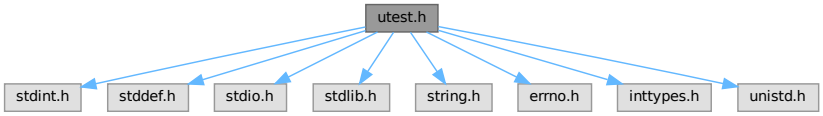
5.8 utest.h File Reference

Single-header unit testing framework by Sean Middleditch.

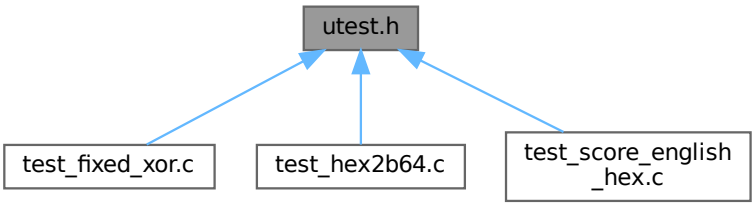
```
#include <stdint.h>
#include <stddef.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <inttypes.h>
#include <unistd.h>
```

Include dependency graph for utest.h:



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



Data Structures

- struct utest_test_state_s
- struct utest_state_s

Macros

- #define UTEST_C_FUNC
- #define UTEST_TEST_PASSED (0)
- #define UTEST_TEST_FAILURE (1)
- #define UTEST_TEST_SKIPPED (2)
- #define UTEST_ATTRIBUTE(a) __attribute__((a))
- #define UTEST_PRId64 PRId64
- #define UTEST_PRIu64 PRIu64
- #define UTEST_INLINE inline
- #define UTEST_INITIALIZER(f)
- #define UTEST_CAST(type, x) ((type)(x))
- #define UTEST_PTR_CAST(type, x) ((type)(x))
- #define UTEST_EXTERN extern
- #define UTEST_NULL 0
- #define UTEST_COLOUR_OUTPUT() (isatty(STDOUT_FILENO))
- #define UTEST_UNUSED UTEST_ATTRIBUTE(unused)
- #define UTEST_PRINTF(...)
- #define UTEST_SNPRINTF(...) snprintf(__VA_ARGS__)

- #define utest_type_printer(...) UTEST_PRINTF("undef")
- #define UTEST_SURPRESS_WARNING_BEGIN
- #define UTEST_SURPRESS_WARNING_END
- #define UTEST_AUTO(x) __typeof__(x + 0)
- #define UTEST_STRNCMP(x, y, size) strncmp(x, y, size)
- #define UTEST_STRNCPY(x, y, size) strncpy(x, y, size)
- #define UTEST_SKIP(msg)
- #define UTEST_COND(x, y, cond, msg, is_assert)
- #define EXPECT_EQ(x, y) UTEST_COND(x, y, ==, "", 0)
- #define EXPECT_EQ_MSG(x, y, msg) UTEST_COND(x, y, ==, msg, 0)
- #define ASSERT_EQ(x, y) UTEST_COND(x, y, ==, "", 1)
- #define ASSERT_EQ_MSG(x, y, msg) UTEST_COND(x, y, ==, msg, 1)
- #define EXPECT_NE(x, y) UTEST_COND(x, y, !=, "", 0)
- #define EXPECT_NE_MSG(x, y, msg) UTEST_COND(x, y, !=, msg, 0)
- #define ASSERT_NE(x, y) UTEST_COND(x, y, !=, "", 1)
- #define ASSERT_NE_MSG(x, y, msg) UTEST_COND(x, y, !=, msg, 1)
- #define EXPECT_LT(x, y) UTEST_COND(x, y, <, "", 0)
- #define EXPECT_LT_MSG(x, y, msg) UTEST_COND(x, y, <, msg, 0)
- #define ASSERT_LT(x, y) UTEST_COND(x, y, <, "", 1)
- #define ASSERT_LT_MSG(x, y, msg) UTEST_COND(x, y, <, msg, 1)
- #define EXPECT_LE(x, y) UTEST_COND(x, y, <=, "", 0)
- #define EXPECT_LE_MSG(x, y, msg) UTEST_COND(x, y, <=, msg, 0)
- #define ASSERT_LE(x, y) UTEST_COND(x, y, <=, "", 1)
- #define ASSERT_LE_MSG(x, y, msg) UTEST_COND(x, y, <=, msg, 1)
- #define EXPECT_GT(x, y) UTEST_COND(x, y, >, "", 0)
- #define EXPECT_GT_MSG(x, y, msg) UTEST_COND(x, y, >, msg, 0)
- #define ASSERT_GT(x, y) UTEST_COND(x, y, >, "", 1)
- #define ASSERT_GT_MSG(x, y, msg) UTEST_COND(x, y, >, msg, 1)
- #define EXPECT_GE(x, y) UTEST_COND(x, y, >=, "", 0)
- #define EXPECT_GE_MSG(x, y, msg) UTEST_COND(x, y, >=, msg, 0)
- #define ASSERT_GE(x, y) UTEST_COND(x, y, >=, "", 1)
- #define ASSERT_GE_MSG(x, y, msg) UTEST_COND(x, y, >=, msg, 1)
- #define UTEST_TRUE(x, msg, is_assert)
- #define EXPECT_TRUE(x) UTEST_TRUE(x, "", 0)
- #define EXPECT_TRUE_MSG(x, msg) UTEST_TRUE(x, msg, 0)
- #define ASSERT_TRUE(x) UTEST_TRUE(x, "", 1)
- #define ASSERT_TRUE_MSG(x, msg) UTEST_TRUE(x, msg, 1)
- #define UTEST_FALSE(x, msg, is_assert)
- #define EXPECT_FALSE(x) UTEST_FALSE(x, "", 0)
- #define EXPECT_FALSE_MSG(x, msg) UTEST_FALSE(x, msg, 0)
- #define ASSERT_FALSE(x) UTEST_FALSE(x, "", 1)
- #define ASSERT_FALSE_MSG(x, msg) UTEST_FALSE(x, msg, 1)
- #define UTEST_STREQ(x, y, msg, is_assert)
- #define EXPECT_STREQ(x, y) UTEST_STREQ(x, y, "", 0)
- #define EXPECT_STREQ_MSG(x, y, msg) UTEST_STREQ(x, y, msg, 0)
- #define ASSERT_STREQ(x, y) UTEST_STREQ(x, y, "", 1)
- #define ASSERT_STREQ_MSG(x, y, msg) UTEST_STREQ(x, y, msg, 1)
- #define UTEST_STRNE(x, y, msg, is_assert)
- #define EXPECT_STRNE(x, y) UTEST_STRNE(x, y, "", 0)
- #define EXPECT_STRNE_MSG(x, y, msg) UTEST_STRNE(x, y, msg, 0)
- #define ASSERT_STRNE(x, y) UTEST_STRNE(x, y, "", 1)
- #define ASSERT_STRNE_MSG(x, y, msg) UTEST_STRNE(x, y, msg, 1)
- #define UTEST_STRNEQ(x, y, n, msg, is_assert)
- #define EXPECT_STRNEQ(x, y, n) UTEST_STRNEQ(x, y, n, "", 0)
- #define EXPECT_STRNEQ_MSG(x, y, n, msg) UTEST_STRNEQ(x, y, n, msg, 0)

- #define ASSERT_STRNEQ(x, y, n) UTEST_STRNEQ(x, y, n, "", 1)
- #define ASSERT_STRNEQ_MSG(x, y, n, msg) UTEST_STRNEQ(x, y, n, msg, 1)
- #define UTEST_STRNNE(x, y, n, msg, is_assert)
- #define EXPECT_STRNNE(x, y, n) UTEST_STRNNE(x, y, n, "", 0)
- #define EXPECT_STRNNE_MSG(x, y, n, msg) UTEST_STRNNE(x, y, n, msg, 0)
- #define ASSERT_STRNNE(x, y, n) UTEST_STRNNE(x, y, n, "", 1)
- #define ASSERT_STRNNE_MSG(x, y, n, msg) UTEST_STRNNE(x, y, n, msg, 1)
- #define UTEST_NEAR(x, y, epsilon, msg, is_assert)
- #define EXPECT_NEAR(x, y, epsilon) UTEST_NEAR(x, y, epsilon, "", 0)
- #define EXPECT_NEAR_MSG(x, y, epsilon, msg) UTEST_NEAR(x, y, epsilon, msg, 0)
- #define ASSERT_NEAR(x, y, epsilon) UTEST_NEAR(x, y, epsilon, "", 1)
- #define ASSERT_NEAR_MSG(x, y, epsilon, msg) UTEST_NEAR(x, y, epsilon, msg, 1)
- #define UTEST_SURPRESS_WARNINGS_BEGIN
- #define UTEST_SURPRESS_WARNINGS_END
- #define UTEST(SET, NAME)
- #define UTEST_F_SETUP(FIXTURE)
- #define UTEST_F_TEARDOWN(FIXTURE)
- #define UTEST_F(FIXTURE, NAME)
- #define UTEST_I_SETUP(FIXTURE)
- #define UTEST_I_TEARDOWN(FIXTURE)
- #define UTEST_I(FIXTURE, NAME, INDEX)
- #define UTEST_STATE() struct utest_state_s utest_state = {0, 0, 0}
- #define UTEST_MAIN()

Typedefs

- typedef int64_t utest_int64_t
- typedef uint64_t utest_uint64_t
- typedef uint32_t utest_uint32_t
- typedef void(∗ utest_testcase_t) (int ∗, size_t)

Functions

- static UTEST_INLINE void ∗ utest_realloc (void ∗const pointer, size_t new_size)
- static UTEST_INLINE utest_int64_t utest_mul_div (const utest_int64_t value, const utest_int64_t numer, const utest_int64_t denom)
- static UTEST_INLINE utest_int64_t utest_ns (void)
- UTEST_WEAK double utest_fabs (double d)
- UTEST_WEAK int utest_isnan (double d)
- UTEST_WEAK int utest_should_filter_test (const char ∗filter, const char ∗testcase)
- static UTEST_INLINE FILE ∗ utest_fopen (const char ∗filename, const char ∗mode)
- static UTEST_INLINE int utest_main (int argc, const char ∗const argv[ ])

Variables

- UTEST_EXTERN struct utest_state_s utest_state

5.8.1 Detailed Description

Single-header unit testing framework by Sean Middleditch.

5.8.2 Macro Definition Documentation

5.8.2.1 ASSERT_EQ

```
#define ASSERT_EQ(
                x,
                y ) UTEST_COND(x, y, ==, "", 1)
```

5.8.2.2 ASSERT_EQ_MSG

```
#define ASSERT_EQ_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, ==, msg, 1)
```

5.8.2.3 ASSERT_FALSE

```
#define ASSERT_FALSE(
              x ) UTEST_FALSE(x, "", 1)
```

5.8.2.4 ASSERT_FALSE_MSG

```
#define ASSERT_FALSE_MSG(
              x,
              msg ) UTEST_FALSE(x, msg, 1)
```

5.8.2.5 ASSERT_GE

```
#define ASSERT_GE(
              x,
              y ) UTEST_COND(x, y, >=, "", 1)
```

5.8.2.6 ASSERT_GE_MSG

```
#define ASSERT_GE_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, >=, msg, 1)
```

5.8.2.7 ASSERT_GT

```
#define ASSERT_GT(
              x,
              y ) UTEST_COND(x, y, >, "", 1)
```

5.8.2.8 ASSERT_GT_MSG

```
#define ASSERT_GT_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, >, msg, 1)
```

5.8.2.9 ASSERT_LE

```
#define ASSERT_LE(
              x,
              y ) UTEST_COND(x, y, <=, "", 1)
```

5.8.2.10 ASSERT_LE_MSG

```
#define ASSERT_LE_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, <=, msg, 1)
```

5.8.2.11 ASSERT_LT

```
#define ASSERT_LT(
              x,
              y ) UTEST_COND(x, y, <, "", 1)
```

5.8.2.12 ASSERT_LT_MSG

```
#define ASSERT_LT_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, <, msg, 1)
```

5.8.2.13 ASSERT_NE

```
#define ASSERT_NE(
              x,
              y ) UTEST_COND(x, y, !=, "", 1)
```

5.8.2.14 ASSERT_NE_MSG

```
#define ASSERT_NE_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, !=, msg, 1)
```

5.8.2.15 ASSERT_NEAR

```
#define ASSERT_NEAR(
              x,
              y,
              epsilon ) UTEST_NEAR(x, y, epsilon, "", 1)
```

5.8.2.16 ASSERT_NEAR_MSG

```
#define ASSERT_NEAR_MSG(
              x,
              y,
              epsilon,
              msg ) UTEST_NEAR(x, y, epsilon, msg, 1)
```

5.8.2.17 ASSERT_STREQ

```
#define ASSERT_STREQ(
              x,
              y ) UTEST_STREQ(x, y, "", 1)
```

5.8.2.18 ASSERT_STREQ_MSG

```
#define ASSERT_STREQ_MSG(
              x,
              y,
              msg ) UTEST_STREQ(x, y, msg, 1)
```

5.8.2.19 ASSERT_STRNE

```
#define ASSERT_STRNE(
              x,
              y ) UTEST_STRNE(x, y, "", 1)
```

5.8.2.20 ASSERT_STRNE_MSG

```
#define ASSERT_STRNE_MSG(
              x,
              y,
              msg ) UTEST_STRNE(x, y, msg, 1)
```

5.8.2.21 ASSERT_STRNEQ

```
#define ASSERT_STRNEQ(
              x,
              y,
              n ) UTEST_STRNEQ(x, y, n, "", 1)
```

5.8.2.22 ASSERT_STRNEQ_MSG

```
#define ASSERT_STRNEQ_MSG(
              x,
              y,
              n,
              msg ) UTEST_STRNEQ(x, y, n, msg, 1)
```

5.8.2.23 ASSERT_STRNNE

```
#define ASSERT_STRNNE(
              x,
              y,
              n ) UTEST_STRNNE(x, y, n, "", 1)
```

5.8.2.24 ASSERT_STRNNE_MSG

```
#define ASSERT_STRNNE_MSG(
              x,
              y,
              n,
              msg ) UTEST_STRNNE(x, y, n, msg, 1)
```

5.8.2.25 ASSERT_TRUE

```
#define ASSERT_TRUE(
              x ) UTEST_TRUE(x, "", 1)
```

5.8.2.26 ASSERT_TRUE_MSG

```
#define ASSERT_TRUE_MSG(
              x,
              msg ) UTEST_TRUE(x, msg, 1)
```

5.8.2.27 EXPECT_EQ

```
#define EXPECT_EQ(
              x,
              y ) UTEST_COND(x, y, ==, "", 0)
```

5.8.2.28 EXPECT_EQ_MSG

```
#define EXPECT_EQ_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, ==, msg, 0)
```

5.8.2.29 EXPECT_FALSE

```
#define EXPECT_FALSE(
              x ) UTEST_FALSE(x, "", 0)
```

5.8.2.30 EXPECT_FALSE_MSG

```
#define EXPECT_FALSE_MSG(
              x,
              msg ) UTEST_FALSE(x, msg, 0)
```

5.8.2.31 EXPECT_GE

```
#define EXPECT_GE(
              x,
              y ) UTEST_COND(x, y, >=, "", 0)
```

5.8.2.32 EXPECT_GE_MSG

```
#define EXPECT_GE_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, >=, msg, 0)
```

5.8.2.33 EXPECT_GT

```
#define EXPECT_GT(
              x,
              y ) UTEST_COND(x, y, >, "", 0)
```

5.8.2.34 EXPECT_GT_MSG

```
#define EXPECT_GT_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, >, msg, 0)
```

5.8.2.35 EXPECT_LE

```
#define EXPECT_LE(
              x,
              y ) UTEST_COND(x, y, <=, "", 0)
```

5.8.2.36 EXPECT_LE_MSG

```
#define EXPECT_LE_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, <=, msg, 0)
```

5.8.2.37 EXPECT_LT

```
#define EXPECT_LT(
              x,
              y ) UTEST_COND(x, y, <, "", 0)
```

5.8.2.38 EXPECT_LT_MSG

```
#define EXPECT_LT_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, <, msg, 0)
```

5.8.2.39 EXPECT_NE

```
#define EXPECT_NE(
              x,
              y ) UTEST_COND(x, y, !=, "", 0)
```

5.8.2.40 EXPECT_NE_MSG

```
#define EXPECT_NE_MSG(
              x,
              y,
              msg ) UTEST_COND(x, y, !=, msg, 0)
```

5.8.2.41 EXPECT_NEAR

```
#define EXPECT_NEAR(
              x,
              y,
              epsilon ) UTEST_NEAR(x, y, epsilon, "", 0)
```

5.8.2.42 EXPECT_NEAR_MSG

```
#define EXPECT_NEAR_MSG(
              x,
              y,
              epsilon,
              msg ) UTEST_NEAR(x, y, epsilon, msg, 0)
```

5.8.2.43 EXPECT_STREQ

```
#define EXPECT_STREQ(
              x,
              y ) UTEST_STREQ(x, y, "", 0)
```

5.8.2.44 EXPECT_STREQ_MSG

```
#define EXPECT_STREQ_MSG(
              x,
              y,
              msg ) UTEST_STREQ(x, y, msg, 0)
```

5.8.2.45 EXPECT_STRNE

```
#define EXPECT_STRNE(
              x,
              y ) UTEST_STRNE(x, y, "", 0)
```

5.8.2.46 EXPECT_STRNE_MSG

```
#define EXPECT_STRNE_MSG(
              x,
              y,
              msg ) UTEST_STRNE(x, y, msg, 0)
```

5.8.2.47 EXPECT_STRNEQ

```
#define EXPECT_STRNEQ(
              x,
              y,
              n ) UTEST_STRNEQ(x, y, n, "", 0)
```

5.8.2.48 EXPECT_STRNEQ_MSG

```
#define EXPECT_STRNEQ_MSG(
              x,
              y,
              n,
              msg ) UTEST_STRNEQ(x, y, n, msg, 0)
```

5.8.2.49 EXPECT_STRNNE

```
#define EXPECT_STRNNE(
              x,
              y,
              n ) UTEST_STRNNE(x, y, n, "", 0)
```

5.8.2.50 EXPECT_STRNNE_MSG

```
#define EXPECT_STRNNE_MSG(
              x,
              y,
              n,
              msg ) UTEST_STRNNE(x, y, n, msg, 0)
```

5.8.2.51 EXPECT_TRUE

```
#define EXPECT_TRUE(
              x ) UTEST_TRUE(x, "", 0)
```

5.8.2.52 EXPECT_TRUE_MSG

```
#define EXPECT_TRUE_MSG(
              x,
              msg ) UTEST_TRUE(x, msg, 0)
```

5.8.2.53 UTEST

```
#define UTEST(
              SET,
              NAME )
```

5.8.2.54 UTEST_ATTRIBUTE

```
#define UTEST_ATTRIBUTE(
              a ) __attribute__((a))
```

5.8.2.55 UTEST_AUTO

```
#define UTEST_AUTO(
              x ) __typeof__(x + 0)
```

5.8.2.56 UTEST_C_FUNC

```
#define UTEST_C_FUNC
```

5.8.2.57 UTEST_CAST

```
#define UTEST_CAST(
              type,
              x ) ((type)(x))
```

5.8.2.58 UTEST_COLOUR_OUTPUT

```
#define UTEST_COLOUR_OUTPUT( ) (isatty(STDOUT_FILENO))
```

5.8.2.59 UTEST_COND

```
#define UTEST_COND(
              x,
              y,
              cond,
              msg,
              is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                          \
    if (!(x)cond(y))) {                                                      \
      UTEST_PRINTF("%s:%i: Failure (Expected " #cond " Actual)", __FILE__,   \
                   __LINE__);                                                \
      if (strlen(msg) > 0) {                                                 \
        UTEST_PRINTF(" Message : %s", msg);                                  \
      }                                                                      \
      UTEST_PRINTF("\n");                                                    \
      *utest_result = UTEST_TEST_FAILURE;                                    \
      if (is_assert) {                                                       \
        return;                                                              \
      }                                                                      \
    }                                                                        \
  }                                                                          \
  while (0)                                                                  \
  UTEST_SURPRESS_WARNING_END
```

5.8.2.60 UTEST_EXTERN

```
#define UTEST_EXTERN extern
```

5.8.2.61 UTEST_F

```
#define UTEST_F(
              FIXTURE,
              NAME )
```

5.8.2.62 UTEST_F_SETUP

```
#define UTEST_F_SETUP(
              FIXTURE )
```

Value:

```
  static void utest_f_setup_##FIXTURE(int *utest_result,                     \
                                      struct FIXTURE *utest_fixture)
```

5.8.2.63 UTEST_F_TEARDOWN

```
#define UTEST_F_TEARDOWN(
              FIXTURE )
```

Value:

```
  static void utest_f_teardown_##FIXTURE(int *utest_result,                  \
                                         struct FIXTURE *utest_fixture)
```

5.8.2.64 UTEST_FALSE

```
#define UTEST_FALSE(
              x,
```

```
              msg,
              is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                          \
    const int xEval = !!(x);                                                 \
    if (xEval) {                                                             \
      UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                  \
      UTEST_PRINTF("  Expected : false\n");                                  \
      UTEST_PRINTF("    Actual : %s\n", (xEval) ? "true" : "false");         \
      if (strlen(msg) > 0) {                                                 \
        UTEST_PRINTF("   Message : %s\n", msg);                              \
      }                                                                      \
      *utest_result = UTEST_TEST_FAILURE;                                    \
      if (is_assert) {                                                       \
        return;                                                              \
      }                                                                      \
    }                                                                        \
  }                                                                          \
  while (0)                                                                  \
  UTEST_SURPRESS_WARNING_END
```

5.8.2.65 UTEST_I

```
#define UTEST_I(
              FIXTURE,
              NAME,
              INDEX )
```

5.8.2.66 UTEST_I_SETUP

```
#define UTEST_I_SETUP(
              FIXTURE )
```

Value:

```
  static void utest_i_setup_##FIXTURE(                                       \
      int *utest_result, struct FIXTURE *utest_fixture, size_t utest_index)
```

5.8.2.67 UTEST_I_TEARDOWN

```
#define UTEST_I_TEARDOWN(
              FIXTURE )
```

Value:

```
  static void utest_i_teardown_##FIXTURE(                                    \
      int *utest_result, struct FIXTURE *utest_fixture, size_t utest_index)
```

5.8.2.68 UTEST_INITIALIZER

```
#define UTEST_INITIALIZER(
              f )
```

Value:

```
  static void f(void) UTEST_ATTRIBUTE(constructor);                          \
  static void f(void)
```

5.8.2.69 UTEST_INLINE

```
#define UTEST_INLINE inline
```

5.8.2.70 UTEST_MAIN

```
#define UTEST_MAIN( )
```

Value:

```
  UTEST_STATE();                                                             \
  int main(int argc, const char *const argv[]) {                             \
    return utest_main(argc, argv);                                           \
  }
```

5.8.2.71 UTEST_NEAR

```
#define UTEST_NEAR(
              x,
```

```
                y,
                epsilon,
                msg,
                is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                                \
    const double diff =                                                            \
        utest_fabs(UTEST_CAST(double, x) - UTEST_CAST(double, y));                 \
    if (diff > UTEST_CAST(double, epsilon) || utest_isnan(diff)) {                 \
      UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                        \
      UTEST_PRINTF("  Expected : %f\n", UTEST_CAST(double, x));                    \
      UTEST_PRINTF("    Actual : %f\n", UTEST_CAST(double, y));                    \
      if (strlen(msg) > 0) {                                                       \
        UTEST_PRINTF("   Message : %s\n", msg);                                    \
      }                                                                            \
      *utest_result = UTEST_TEST_FAILURE;                                          \
      if (is_assert) {                                                             \
        return;                                                                    \
      }                                                                            \
    }                                                                              \
  }                                                                                \
  while (0)                                                                        \
  UTEST_SURPRESS_WARNING_END
```

5.8.2.72 UTEST_NULL

```
#define UTEST_NULL 0
```

5.8.2.73 UTEST_PRId64

```
#define UTEST_PRId64 PRId64
```

5.8.2.74 UTEST_PRINTF

```
#define UTEST_PRINTF(
                ...  )
```

Value:

```
  if (utest_state.output) {                                                        \
    fprintf(utest_state.output, __VA_ARGS__);                                      \
  }                                                                                \
  printf(__VA_ARGS__)
```

5.8.2.75 UTEST_PRIu64

```
#define UTEST_PRIu64 PRIu64
```

5.8.2.76 UTEST_PTR_CAST

```
#define UTEST_PTR_CAST(
                type,
                x ) ((type)(x))
```

5.8.2.77 UTEST_SKIP

```
#define UTEST_SKIP(
                msg )
```

Value:

```
  do {                                                                             \
    UTEST_PRINTF("   Skipped : '%s'\n", (msg));                                    \
    *utest_result = UTEST_TEST_SKIPPED;                                            \
    return;                                                                        \
  } while (0)
```

5.8.2.78 UTEST_SNPRINTF

```
#define UTEST_SNPRINTF(
                ...  ) snprintf(__VA_ARGS__)
```

5.8.2.79 UTEST_STATE

```
#define UTEST_STATE( ) struct utest_state_s utest_state = {0, 0, 0}
```

5.8.2.80 UTEST_STREQ

```
#define UTEST_STREQ(
              x,
              y,
              msg,
              is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                          \
    const char *xEval = (x);                                                 \
    const char *yEval = (y);                                                 \
    if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                        \
        0 != strcmp(xEval, yEval)) {                                         \
      UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                  \
      UTEST_PRINTF("  Expected : \"%s\"\n", xEval);                         \
      UTEST_PRINTF("    Actual : \"%s\"\n", yEval);                         \
      if (strlen(msg) > 0) {                                                 \
        UTEST_PRINTF("   Message : %s\n", msg);                              \
      }                                                                      \
      *utest_result = UTEST_TEST_FAILURE;                                    \
      if (is_assert) {                                                       \
        return;                                                              \
      }                                                                      \
    }                                                                        \
  }                                                                          \
  while (0)                                                                  \
  UTEST_SURPRESS_WARNING_END
```

5.8.2.81 UTEST_STRNCMP

```
#define UTEST_STRNCMP(
              x,
              y,
              size ) strncmp(x, y, size)
```

5.8.2.82 UTEST_STRNCPY

```
#define UTEST_STRNCPY(
              x,
              y,
              size ) strncpy(x, y, size)
```

5.8.2.83 UTEST_STRNE

```
#define UTEST_STRNE(
              x,
              y,
              msg,
              is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                          \
    const char *xEval = (x);                                                 \
    const char *yEval = (y);                                                 \
    if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                        \
        0 == strcmp(xEval, yEval)) {                                         \
      UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                  \
      UTEST_PRINTF("  Expected : \"%s\"\n", xEval);                         \
      UTEST_PRINTF("    Actual : \"%s\"\n", yEval);                         \
      if (strlen(msg) > 0) {                                                 \
        UTEST_PRINTF("   Message : %s\n", msg);                              \
      }                                                                      \
      *utest_result = UTEST_TEST_FAILURE;                                    \
      if (is_assert) {                                                       \
        return;                                                              \
      }                                                                      \
    }                                                                        \
  }                                                                          \
  while (0)                                                                  \
```

```
  UTEST_SURPRESS_WARNING_END
```

5.8.2.84 UTEST_STRNEQ

```
#define UTEST_STRNEQ(
                x,
                y,
                n,
                msg,
                is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                                 \
    const char *xEval = (x);                                                        \
    const char *yEval = (y);                                                        \
    const size_t nEval = UTEST_CAST(size_t, n);                                     \
    if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                               \
        0 != UTEST_STRNCMP(xEval, yEval, nEval)) {                                  \
      UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                         \
      UTEST_PRINTF("  Expected : \"%.*s\"\n", UTEST_CAST(int, nEval), xEval);       \
      UTEST_PRINTF("    Actual : \"%.*s\"\n", UTEST_CAST(int, nEval), yEval);       \
      if (strlen(msg) > 0) {                                                        \
        UTEST_PRINTF("   Message : %s\n", msg);                                     \
      }                                                                             \
      *utest_result = UTEST_TEST_FAILURE;                                           \
      if (is_assert) {                                                              \
        return;                                                                     \
      }                                                                             \
    }                                                                               \
  }                                                                                 \
  while (0)                                                                         \
  UTEST_SURPRESS_WARNING_END
```

5.8.2.85 UTEST_STRNNE

```
#define UTEST_STRNNE(
                x,
                y,
                n,
                msg,
                is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                                 \
    const char *xEval = (x);                                                        \
    const char *yEval = (y);                                                        \
    const size_t nEval = UTEST_CAST(size_t, n);                                     \
    if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                               \
        0 == UTEST_STRNCMP(xEval, yEval, nEval)) {                                  \
      UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                         \
      UTEST_PRINTF("  Expected : \"%.*s\"\n", UTEST_CAST(int, nEval), xEval);       \
      UTEST_PRINTF("    Actual : \"%.*s\"\n", UTEST_CAST(int, nEval), yEval);       \
      if (strlen(msg) > 0) {                                                        \
        UTEST_PRINTF("   Message : %s\n", msg);                                     \
      }                                                                             \
      *utest_result = UTEST_TEST_FAILURE;                                           \
      if (is_assert) {                                                              \
        return;                                                                     \
      }                                                                             \
    }                                                                               \
  }                                                                                 \
  while (0)                                                                         \
  UTEST_SURPRESS_WARNING_END
```

5.8.2.86 UTEST_SURPRESS_WARNING_BEGIN

```
#define UTEST_SURPRESS_WARNING_BEGIN
```

5.8.2.87 UTEST_SURPRESS_WARNING_END

```
#define UTEST_SURPRESS_WARNING_END
```

5.8.2.88 UTEST_SURPRESS_WARNINGS_BEGIN

```
#define UTEST_SURPRESS_WARNINGS_BEGIN
```

5.8.2.89 UTEST_SURPRESS_WARNINGS_END

```
#define UTEST_SURPRESS_WARNINGS_END
```

5.8.2.90 UTEST_TEST_FAILURE

```
#define UTEST_TEST_FAILURE (1)
```

5.8.2.91 UTEST_TEST_PASSED

```
#define UTEST_TEST_PASSED (0)
```

5.8.2.92 UTEST_TEST_SKIPPED

```
#define UTEST_TEST_SKIPPED (2)
```

5.8.2.93 UTEST_TRUE

```
#define UTEST_TRUE(
              x,
              msg,
              is_assert )
```

Value:

```
  UTEST_SURPRESS_WARNING_BEGIN do {                                          \
    const int xEval = !!(x);                                                 \
    if (!(xEval)) {                                                          \
      UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                  \
      UTEST_PRINTF("  Expected : true\n");                                   \
      UTEST_PRINTF("    Actual : %s\n", (xEval) ? "true" : "false");         \
      if (strlen(msg) > 0) {                                                 \
        UTEST_PRINTF("   Message : %s\n", msg);                              \
      }                                                                      \
      *utest_result = UTEST_TEST_FAILURE;                                    \
      if (is_assert) {                                                       \
        return;                                                              \
      }                                                                      \
    }                                                                        \
  }                                                                          \
  while (0)                                                                  \
  UTEST_SURPRESS_WARNING_END
```

5.8.2.94 utest_type_printer

```
#define utest_type_printer(
              ... ) UTEST_PRINTF("undef")
```

5.8.2.95 UTEST_UNUSED

```
#define UTEST_UNUSED UTEST_ATTRIBUTE(unused)
```

5.8.3 Typedef Documentation

5.8.3.1 utest_int64_t

```
typedef int64_t utest_int64_t
```

5.8.3.2 utest_testcase_t

```
typedef void(* utest_testcase_t) (int *, size_t)
```

5.8.3.3 utest_uint32_t

```
typedef uint32_t utest_uint32_t
```

5.8.3.4 utest_uint64_t

```
typedef uint64_t utest_uint64_t
```

5.8.4 Function Documentation

5.8.4.1 utest_fabs()

```
UTEST_WEAK double utest_fabs (
            double d )
```

5.8.4.2 utest_fopen()

```
static UTEST_INLINE FILE * utest_fopen (
            const char * filename,
            const char * mode )  [static]
```

5.8.4.3 utest_isnan()

```
UTEST_WEAK int utest_isnan (
            double d )
```

5.8.4.4 utest_main()

```
int utest_main (
            int argc,
            const char *const argv[] )  [static]
```

5.8.4.5 utest_mul_div()

```
static UTEST_INLINE utest_int64_t utest_mul_div (
            const utest_int64_t value,
            const utest_int64_t numer,
            const utest_int64_t denom )  [static]
```

5.8.4.6 utest_ns()

```
static UTEST_INLINE utest_int64_t utest_ns (
            void  )  [static]
```

5.8.4.7 utest_realloc()

```
static UTEST_INLINE void * utest_realloc (
            void *const pointer,
            size_t new_size )  [static]
```

5.8.4.8 utest_should_filter_test()

```
UTEST_WEAK int utest_should_filter_test (
            const char * filter,
            const char * testcase )
```

5.8.5 Variable Documentation

5.8.5.1 utest_state

```
UTEST_EXTERN struct utest_state_s utest_state
```

5.9 utest.h

Go to the documentation of this file.

```
00001 /*
00002    The latest version of this library is available on GitHub;
00003    https://github.com/sheredom/utest.h
00004 */
```

```
00005
00006 /*
00007    This is free and unencumbered software released into the public domain.
00008
00009    Anyone is free to copy, modify, publish, use, compile, sell, or
00010    distribute this software, either in source code form or as a compiled
00011    binary, for any purpose, commercial or non-commercial, and by any
00012    means.
00013
00014    In jurisdictions that recognize copyright laws, the author or authors
00015    of this software dedicate any and all copyright interest in the
00016    software to the public domain. We make this dedication for the benefit
00017    of the public at large and to the detriment of our heirs and
00018    successors. We intend this dedication to be an overt act of
00019    relinquishment in perpetuity of all present and future rights to this
00020    software under copyright law.
00021
00022    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
00023    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
00024    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
00025    IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
00026    OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
00027    ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
00028    OTHER DEALINGS IN THE SOFTWARE.
00029
00030    For more information, please refer to <http://unlicense.org/>
00031 */
00032
00038 #ifndef SHEREDOM_UTEST_H_INCLUDED
00039 #define SHEREDOM_UTEST_H_INCLUDED
00040
00041 #ifdef _MSC_VER
00042 /*
00043    Disable warning about not inlining 'inline' functions.
00044 */
00045 #pragma warning(disable : 4710)
00046
00047 /*
00048    Disable warning about inlining functions that are not marked 'inline'.
00049 */
00050 #pragma warning(disable : 4711)
00051
00052 /*
00053    Disable warning for alignment padding added
00054 */
00055 #pragma warning(disable : 4820)
00056
00057 #if _MSC_VER > 1900
00058 /*
00059   Disable warning about preprocessor macros not being defined in MSVC headers.
00060 */
00061 #pragma warning(disable : 4668)
00062
00063 /*
00064   Disable warning about no function prototype given in MSVC headers.
00065 */
00066 #pragma warning(disable : 4255)
00067
00068 /*
00069   Disable warning about pointer or reference to potentially throwing function.
00070 */
00071 #pragma warning(disable : 5039)
00072
00073 /*
00074   Disable warning about macro expansion producing 'defined' has undefined
00075   behavior.
00076 */
00077 #pragma warning(disable : 5105)
00078 #endif
00079
00080 #if _MSC_VER > 1930
00081 /*
00082   Disable warning about 'const' variable is not used.
00083 */
00084 #pragma warning(disable : 5264)
00085 #endif
00086
00087 #pragma warning(push, 1)
00088 #endif
00089
00090 #if defined(_MSC_VER) && (_MSC_VER < 1920)
00091 typedef __int64 utest_int64_t;
00092 typedef unsigned __int64 utest_uint64_t;
00093 typedef unsigned __int32 utest_uint32_t;
00094 #else
00095 #include <stdint.h>
00096 typedef int64_t utest_int64_t;
```

```
00097 typedef uint64_t utest_uint64_t;
00098 typedef uint32_t utest_uint32_t;
00099 #endif
00100
00101 #include <stddef.h>
00102 #include <stdio.h>
00103 #include <stdlib.h>
00104 #include <string.h>
00105 #include <errno.h>
00106
00107 #if defined(__cplusplus)
00108 #if defined(_MSC_VER) && !defined(_CPPUNWIND)
00109 /* We're on MSVC and the compiler is compiling without exception support! */
00110 #elif !defined(_MSC_VER) && !defined(__EXCEPTIONS)
00111 /* We're on a GCC/Clang compiler that doesn't have exception support! */
00112 #else
00113 #define UTEST_HAS_EXCEPTIONS 1
00114 #endif
00115 #endif
00116
00117 #if defined(UTEST_HAS_EXCEPTIONS)
00118 #include <stdexcept>
00119 #endif
00120
00121 #if defined(_MSC_VER)
00122 #pragma warning(pop)
00123 #endif
00124
00125 #if defined(__cplusplus)
00126 #define UTEST_C_FUNC extern "C"
00127 #else
00128 #define UTEST_C_FUNC
00129 #endif
00130
00131 #define UTEST_TEST_PASSED (0)
00132 #define UTEST_TEST_FAILURE (1)
00133 #define UTEST_TEST_SKIPPED (2)
00134
00135 #if defined(__TINYC__)
00136 #define UTEST_ATTRIBUTE(a) __attribute((a))
00137 #else
00138 #define UTEST_ATTRIBUTE(a) __attribute__((a))
00139 #endif
00140
00141 #if defined(_MSC_VER) || defined(__MINGW64__) || defined(__MINGW32__)
00142
00143 #if defined(__MINGW64__) || defined(__MINGW32__)
00144 #pragma GCC diagnostic push
00145 #pragma GCC diagnostic ignored "-Wpragmas"
00146 #pragma GCC diagnostic ignored "-Wunknown-pragmas"
00147 #endif
00148
00149 #if defined(_WINDOWS_) || defined(_WINDOWS_H)
00150 typedef LARGE_INTEGER utest_large_integer;
00151 #else
00152 // use old QueryPerformanceCounter definitions (not sure is this needed in some
00153 // edge cases or not) on Win7 with VS2015 these extern declaration cause "second
00154 // C linkage of overloaded function not allowed" error
00155 typedef union {
00156   struct {
00157     unsigned long LowPart;
00158     long HighPart;
00159   } DUMMYSTRUCTNAME;
00160   struct {
00161     unsigned long LowPart;
00162     long HighPart;
00163   } u;
00164   utest_int64_t QuadPart;
00165 } utest_large_integer;
00166
00167 UTEST_C_FUNC __declspec(dllimport) int __stdcall QueryPerformanceCounter(
00168     utest_large_integer *);
00169 UTEST_C_FUNC __declspec(dllimport) int __stdcall QueryPerformanceFrequency(
00170     utest_large_integer *);
00171
00172 #if defined(__MINGW64__) || defined(__MINGW32__)
00173 #pragma GCC diagnostic pop
00174 #endif
00175 #endif
00176
00177 #elif defined(__linux__) || defined(__FreeBSD__) || defined(__OpenBSD__) ||     \
00178     defined(__NetBSD__) || defined(__DragonFly__) || defined(__sun__) ||       \
00179     defined(__HAIKU__)
00180 /*
00181    slightly obscure include here - we need to include glibc's features.h, but
00182    we don't want to just include a header that might not be defined for other
00183    c libraries like musl. Instead we include limits.h, which we know on all
```

```
00184    glibc distributions includes features.h
00185 */
00186 #include <limits.h>
00187
00188 #if defined(__GLIBC__) && defined(__GLIBC_MINOR__)
00189 #include <time.h>
00190
00191 #if ((2 < __GLIBC__) || ((2 == __GLIBC__) && (17 <= __GLIBC_MINOR__)))
00192 /* glibc is version 2.17 or above, so we can just use clock_gettime */
00193 #define UTEST_USE_CLOCKGETTIME
00194 #else
00195 #include <sys/syscall.h>
00196 #include <unistd.h>
00197 #endif
00198 #else // Other libc implementations
00199 #include <time.h>
00200 #define UTEST_USE_CLOCKGETTIME
00201 #endif
00202
00203 #elif defined(__APPLE__)
00204 #include <time.h>
00205 #endif
00206
00207 #if defined(_MSC_VER) && (_MSC_VER < 1920)
00208 #define UTEST_PRId64 "I64d"
00209 #define UTEST_PRIu64 "I64u"
00210 #else
00211 #include <inttypes.h>
00212
00213 #define UTEST_PRId64 PRId64
00214 #define UTEST_PRIu64 PRIu64
00215 #endif
00216
00217 #if defined(__cplusplus)
00218 #define UTEST_INLINE inline
00219
00220 #if defined(__clang__)
00221 #define UTEST_INITIALIZER_BEGIN_DISABLE_WARNINGS                                \
00222   _Pragma("clang diagnostic push")                                              \
00223       _Pragma("clang diagnostic ignored \"-Wglobal-constructors\"")
00224
00225 #define UTEST_INITIALIZER_END_DISABLE_WARNINGS _Pragma("clang diagnostic pop")
00226 #else
00227 #define UTEST_INITIALIZER_BEGIN_DISABLE_WARNINGS
00228 #define UTEST_INITIALIZER_END_DISABLE_WARNINGS
00229 #endif
00230
00231 #define UTEST_INITIALIZER(f)                                                    \
00232   struct f##_cpp_struct {                                                       \
00233     f##_cpp_struct();                                                           \
00234   };                                                                            \
00235   UTEST_INITIALIZER_BEGIN_DISABLE_WARNINGS static f##_cpp_struct                \
00236       f##_cpp_global UTEST_INITIALIZER_END_DISABLE_WARNINGS;                    \
00237   f##_cpp_struct::f##_cpp_struct()
00238 #elif defined(_MSC_VER)
00239 #define UTEST_INLINE __forceinline
00240
00241 #if defined(_WIN64)
00242 #define UTEST_SYMBOL_PREFIX
00243 #else
00244 #define UTEST_SYMBOL_PREFIX "_"
00245 #endif
00246
00247 #if defined(__clang__)
00248 #define UTEST_INITIALIZER_BEGIN_DISABLE_WARNINGS                                \
00249   _Pragma("clang diagnostic push")                                              \
00250       _Pragma("clang diagnostic ignored \"-Wmissing-variable-declarations\"")
00251
00252 #define UTEST_INITIALIZER_END_DISABLE_WARNINGS _Pragma("clang diagnostic pop")
00253 #else
00254 #define UTEST_INITIALIZER_BEGIN_DISABLE_WARNINGS
00255 #define UTEST_INITIALIZER_END_DISABLE_WARNINGS
00256 #endif
00257
00258 #pragma section(".CRT$XCU", read)
00259 #define UTEST_INITIALIZER(f)                                                    \
00260   static void __cdecl f(void);                                                  \
00261   UTEST_INITIALIZER_BEGIN_DISABLE_WARNINGS                                      \
00262   __pragma(comment(linker, "/include:" UTEST_SYMBOL_PREFIX #f "_"))             \
00263       UTEST_C_FUNC                                                              \
00264       __declspec(allocate(".CRT$XCU")) void(__cdecl * f##_)(void) = f;          \
00265   UTEST_INITIALIZER_END_DISABLE_WARNINGS                                        \
00266   static void __cdecl f(void)
00267 #else
00268 #if defined(__linux__)
00269 #if defined(__clang__)
00270 #if __has_warning("-Wreserved-id-macro")
```

```
00271 #pragma clang diagnostic push
00272 #pragma clang diagnostic ignored "-Wreserved-id-macro"
00273 #endif
00274 #endif
00275
00276 #define __STDC_FORMAT_MACROS 1
00277
00278 #if defined(__clang__)
00279 #if __has_warning("-Wreserved-id-macro")
00280 #pragma clang diagnostic pop
00281 #endif
00282 #endif
00283 #endif
00284
00285 #define UTEST_INLINE inline
00286
00287 #define UTEST_INITIALIZER(f)                                                \
00288   static void f(void) UTEST_ATTRIBUTE(constructor);                         \
00289   static void f(void)
00290 #endif
00291
00292 #if defined(__cplusplus)
00293 #define UTEST_CAST(type, x) static_cast<type>(x)
00294 #define UTEST_PTR_CAST(type, x) reinterpret_cast<type>(x)
00295 #define UTEST_EXTERN extern "C"
00296 #define UTEST_NULL NULL
00297 #else
00298 #define UTEST_CAST(type, x) ((type)(x))
00299 #define UTEST_PTR_CAST(type, x) ((type)(x))
00300 #define UTEST_EXTERN extern
00301 #define UTEST_NULL 0
00302 #endif
00303
00304 #ifdef _MSC_VER
00305 /*
00306     io.h contains definitions for some structures with natural padding. This is
00307     uninteresting, but for some reason MSVC's behaviour is to warn about
00308     including this system header. That *is* interesting
00309 */
00310 #pragma warning(disable : 4820)
00311 #pragma warning(push, 1)
00312 #include <io.h>
00313 #pragma warning(pop)
00314 #define UTEST_COLOUR_OUTPUT() (_isatty(_fileno(stdout)))
00315 #else
00316 #if defined(__EMSCRIPTEN__)
00317 #include <emscripten/html5.h>
00318 #define UTEST_COLOUR_OUTPUT() false
00319 #else
00320 #include <unistd.h>
00321 #define UTEST_COLOUR_OUTPUT() (isatty(STDOUT_FILENO))
00322 #endif
00323 #endif
00324
00325 static UTEST_INLINE void *utest_realloc(void *const pointer, size_t new_size) {
00326   void *const new_pointer = realloc(pointer, new_size);
00327
00328   if (UTEST_NULL == new_pointer) {
00329     free(pointer);
00330   }
00331
00332   return new_pointer;
00333 }
00334
00335 // Prevent 64-bit integer overflow when computing a timestamp by using a trick
00336 // from Sokol:
00337 // https://github.com/floooh/sokol/blob/189843bf4f86969ca4cc4b6d94e793a37c5128a7/sokol_time.h#L204
00338 static UTEST_INLINE utest_int64_t utest_mul_div(const utest_int64_t value,
00339                                                 const utest_int64_t numer,
00340                                                 const utest_int64_t denom) {
00341   const utest_int64_t q = value / denom;
00342   const utest_int64_t r = value % denom;
00343   return q * numer + r * numer / denom;
00344 }
00345
00346 static UTEST_INLINE utest_int64_t utest_ns(void) {
00347 #if defined(_MSC_VER) || defined(__MINGW64__) || defined(__MINGW32__)
00348   utest_large_integer counter;
00349   utest_large_integer frequency;
00350   QueryPerformanceCounter(&counter);
00351   QueryPerformanceFrequency(&frequency);
00352   return utest_mul_div(counter.QuadPart, 1000000000, frequency.QuadPart);
00353 #elif defined(__linux__) && defined(__STRICT_ANSI__)
00354   return utest_mul_div(clock(), 1000000000, CLOCKS_PER_SEC);
00355 #elif defined(__linux__) || defined(__FreeBSD__) || defined(__OpenBSD__) ||    \
00356     defined(__NetBSD__) || defined(__DragonFly__) || defined(__sun__) ||       \
00357     defined(__HAIKU__)
```

```
00358   struct timespec ts;
00359 #if defined(__STDC_VERSION__) && (__STDC_VERSION__ >= 201112L) &&             \
00360     !defined(__HAIKU__)
00361   timespec_get(&ts, TIME_UTC);
00362 #else
00363   const clockid_t cid = CLOCK_REALTIME;
00364 #if defined(UTEST_USE_CLOCKGETTIME)
00365   clock_gettime(cid, &ts);
00366 #else
00367   syscall(SYS_clock_gettime, cid, &ts);
00368 #endif
00369 #endif
00370   return UTEST_CAST(utest_int64_t, ts.tv_sec) * 1000 * 1000 * 1000 + ts.tv_nsec;
00371 #elif __APPLE__
00372   return UTEST_CAST(utest_int64_t, clock_gettime_nsec_np(CLOCK_UPTIME_RAW));
00373 #elif __EMSCRIPTEN__
00374   return emscripten_performance_now() * 1000000.0;
00375 #else
00376 #error Unsupported platform!
00377 #endif
00378 }
00379
00380 typedef void (*utest_testcase_t)(int *, size_t);
00381
00382 struct utest_test_state_s {
00383   utest_testcase_t func;
00384   size_t index;
00385   char *name;
00386 };
00387
00388 struct utest_state_s {
00389   struct utest_test_state_s *tests;
00390   size_t tests_length;
00391   FILE *output;
00392 };
00393
00394 /* extern to the global state utest needs to execute */
00395 UTEST_EXTERN struct utest_state_s utest_state;
00396
00397 #if defined(_MSC_VER)
00398 #define UTEST_WEAK __forceinline
00399 #elif defined(__MINGW32__) || defined(__MINGW64__)
00400 #define UTEST_WEAK static UTEST_ATTRIBUTE(used)
00401 #elif defined(__clang__) || defined(__GNUC__) || defined(__TINYC__)
00402 #define UTEST_WEAK UTEST_ATTRIBUTE(weak)
00403 #else
00404 #error Non clang, non gcc, non MSVC, non tcc compiler found!
00405 #endif
00406
00407 #if defined(_MSC_VER)
00408 #define UTEST_UNUSED
00409 #else
00410 #define UTEST_UNUSED UTEST_ATTRIBUTE(unused)
00411 #endif
00412
00413 #ifdef __clang__
00414 #pragma clang diagnostic push
00415 #pragma clang diagnostic ignored "-Wvariadic-macros"
00416 #pragma clang diagnostic ignored "-Wc++98-compat-pedantic"
00417 #endif
00418 #define UTEST_PRINTF(...)                                                     \
00419   if (utest_state.output) {                                                   \
00420     fprintf(utest_state.output, __VA_ARGS__);                                 \
00421   }                                                                           \
00422   printf(__VA_ARGS__)
00423 #ifdef __clang__
00424 #pragma clang diagnostic pop
00425 #endif
00426
00427 #ifdef __clang__
00428 #pragma clang diagnostic push
00429 #pragma clang diagnostic ignored "-Wvariadic-macros"
00430 #pragma clang diagnostic ignored "-Wc++98-compat-pedantic"
00431 #endif
00432
00433 #ifdef _MSC_VER
00434 #define UTEST_SNPRINTF(BUFFER, N, ...) _snprintf_s(BUFFER, N, N, __VA_ARGS__)
00435 #else
00436 #define UTEST_SNPRINTF(...) snprintf(__VA_ARGS__)
00437 #endif
00438
00439 #ifdef __clang__
00440 #pragma clang diagnostic pop
00441 #endif
00442
00443 #if defined(__cplusplus)
00444 /* if we are using c++ we can use overloaded methods (its in the language) */
```

```
00445 #define UTEST_OVERLOADABLE
00446 #elif defined(__clang__)
00447 /* otherwise, if we are using clang with c - use the overloadable attribute */
00448 #define UTEST_OVERLOADABLE UTEST_ATTRIBUTE(overloadable)
00449 #endif
00450
00451 #if defined(__cplusplus) && (__cplusplus >= 201103L)
00452
00453 #ifdef __clang__
00454 #pragma clang diagnostic push
00455 #pragma clang diagnostic ignored "-Wc++98-compat-pedantic"
00456 #endif
00457
00458 #include <type_traits>
00459
00460 template <typename T, bool is_enum = std::is_enum<T>::value>
00461 struct utest_type_deducer final {
00462   static void _(const T t);
00463 };
00464
00465 template <> struct utest_type_deducer<char, false> {
00466   static void _(const char c) {
00467     if (std::is_signed<decltype(c)>::value) {
00468       UTEST_PRINTF("%d", static_cast<int>(c));
00469     } else {
00470       UTEST_PRINTF("%u", static_cast<unsigned int>(c));
00471     }
00472   }
00473 };
00474 template <> struct utest_type_deducer<signed char, false> {
00475   static void _(const signed char c) {
00476     UTEST_PRINTF("%d", static_cast<int>(c));
00477   }
00478 };
00479
00480 template <> struct utest_type_deducer<unsigned char, false> {
00481   static void _(const unsigned char c) {
00482     UTEST_PRINTF("%u", static_cast<unsigned int>(c));
00483   }
00484 };
00485
00486 template <> struct utest_type_deducer<short, false> {
00487   static void _(const short s) { UTEST_PRINTF("%d", static_cast<int>(s)); }
00488 };
00489
00490 template <> struct utest_type_deducer<unsigned short, false> {
00491   static void _(const unsigned short s) {
00492     UTEST_PRINTF("%u", static_cast<unsigned>(s));
00493   }
00494 };
00495
00496 template <> struct utest_type_deducer<float, false> {
00497   static void _(const float f) { UTEST_PRINTF("%f", static_cast<double>(f)); }
00498 };
00499
00500 template <> struct utest_type_deducer<double, false> {
00501   static void _(const double d) { UTEST_PRINTF("%f", d); }
00502 };
00503
00504 template <> struct utest_type_deducer<long double, false> {
00505   static void _(const long double d) {
00506 #if defined(__MINGW32__) || defined(__MINGW64__)
00507     /* MINGW is weird - doesn't like LF at all?! */
00508     UTEST_PRINTF("%f", (double)d);
00509 #else
00510     UTEST_PRINTF("%Lf", d);
00511 #endif
00512   }
00513 };
00514
00515 template <> struct utest_type_deducer<int, false> {
00516   static void _(const int i) { UTEST_PRINTF("%d", i); }
00517 };
00518
00519 template <> struct utest_type_deducer<unsigned int, false> {
00520   static void _(const unsigned int i) { UTEST_PRINTF("%u", i); }
00521 };
00522
00523 template <> struct utest_type_deducer<long, false> {
00524   static void _(const long i) { UTEST_PRINTF("%ld", i); }
00525 };
00526
00527 template <> struct utest_type_deducer<unsigned long, false> {
00528   static void _(const unsigned long i) { UTEST_PRINTF("%lu", i); }
00529 };
00530
00531 template <> struct utest_type_deducer<long long, false> {
```

```
00532   static void _(const long long i) { UTEST_PRINTF("%lld", i); }
00533 };
00534
00535 template <> struct utest_type_deducer<unsigned long long, false> {
00536   static void _(const unsigned long long i) { UTEST_PRINTF("%llu", i); }
00537 };
00538
00539 template <> struct utest_type_deducer<bool, false> {
00540   static void _(const bool i) { UTEST_PRINTF(i ? "true" : "false"); }
00541 };
00542
00543 template <typename T> struct utest_type_deducer<const T *, false> {
00544   static void _(const T *t) {
00545     UTEST_PRINTF("%p", static_cast<const void *>(t));
00546   }
00547 };
00548
00549 template <typename T> struct utest_type_deducer<T *, false> {
00550   static void _(T *t) { UTEST_PRINTF("%p", static_cast<void *>(t)); }
00551 };
00552
00553 template <typename T> struct utest_type_deducer<T, true> {
00554   static void _(const T t) {
00555     UTEST_PRINTF("%llu", static_cast<unsigned long long>(t));
00556   }
00557 };
00558
00559 // default printer for all other objects (specialize for custom printing)
00560 template <typename T> struct utest_type_deducer<T, false> {
00561   static void _(const T& t) {
00562     UTEST_PRINTF("(object %p)", static_cast<const void*>(&t));
00563   }
00564 };
00565
00566 template <> struct utest_type_deducer<std::nullptr_t, false> {
00567   static void _(std::nullptr_t t) {
00568     UTEST_PRINTF("%p", static_cast<void *>(t));
00569   }
00570 };
00571
00572 template <typename T>
00573 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(const T& t) {
00574   utest_type_deducer<T>::_(t);
00575 }
00576
00577 #ifdef __clang__
00578 #pragma clang diagnostic pop
00579 #endif
00580
00581 #elif defined(UTEST_OVERLOADABLE)
00582
00583 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(signed char c);
00584 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(signed char c) {
00585   UTEST_PRINTF("%d", UTEST_CAST(int, c));
00586 }
00587
00588 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(unsigned char c);
00589 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(unsigned char c) {
00590   UTEST_PRINTF("%u", UTEST_CAST(unsigned int, c));
00591 }
00592
00593 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(float f);
00594 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(float f) {
00595   UTEST_PRINTF("%f", UTEST_CAST(double, f));
00596 }
00597
00598 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(double d);
00599 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(double d) {
00600   UTEST_PRINTF("%f", d);
00601 }
00602
00603 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long double d);
00604 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long double d) {
00605 #if defined(__MINGW32__) || defined(__MINGW64__)
00606   /* MINGW is weird - doesn't like LF at all?! */
00607   UTEST_PRINTF("%f", (double)d);
00608 #else
00609   UTEST_PRINTF("%Lf", d);
00610 #endif
00611 }
00612
00613 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(int i);
00614 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(int i) {
00615   UTEST_PRINTF("%d", i);
00616 }
00617
00618 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(unsigned int i);
```

```
00619 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(unsigned int i) {
00620   UTEST_PRINTF("%u", i);
00621 }
00622
00623 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long int i);
00624 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long int i) {
00625   UTEST_PRINTF("%ld", i);
00626 }
00627
00628 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long unsigned int i);
00629 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long unsigned int i) {
00630   UTEST_PRINTF("%lu", i);
00631 }
00632
00633 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(const void *p);
00634 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(const void *p) {
00635   UTEST_PRINTF("%p", p);
00636 }
00637
00638 /*
00639    long long is a c++11 extension
00640 */
00641 #if defined(__STDC_VERSION__) && (__STDC_VERSION__ >= 199901L) ||              \
00642     defined(__cplusplus) && (__cplusplus >= 201103L) ||                        \
00643     (defined(__MINGW32__) || defined(__MINGW64__))
00644
00645 #ifdef __clang__
00646 #pragma clang diagnostic push
00647 #pragma clang diagnostic ignored "-Wc++98-compat-pedantic"
00648 #endif
00649
00650 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long long int i);
00651 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long long int i) {
00652   UTEST_PRINTF("%lld", i);
00653 }
00654
00655 UTEST_WEAK UTEST_OVERLOADABLE void utest_type_printer(long long unsigned int i);
00656 UTEST_WEAK UTEST_OVERLOADABLE void
00657 utest_type_printer(long long unsigned int i) {
00658   UTEST_PRINTF("%llu", i);
00659 }
00660
00661 #ifdef __clang__
00662 #pragma clang diagnostic pop
00663 #endif
00664
00665 #endif
00666 #elif defined(__STDC_VERSION__) && (__STDC_VERSION__ >= 201112L) &&            \
00667         !(defined(__MINGW32__) || defined(__MINGW64__)) ||                     \
00668     defined(__TINYC__)
00669 #define utest_type_printer(val)                                                \
00670   UTEST_PRINTF(                                                                \
00671       _Generic((val),                                                          \
00672       signed char: "%d",                                                       \
00673       unsigned char: "%u",                                                     \
00674       short: "%d",                                                             \
00675       unsigned short: "%u",                                                    \
00676       int: "%d",                                                               \
00677       long: "%ld",                                                             \
00678       long long: "%lld",                                                       \
00679       unsigned: "%u",                                                          \
00680       unsigned long: "%lu",                                                    \
00681       unsigned long long: "%llu",                                              \
00682       float: "%f",                                                             \
00683       double: "%f",                                                            \
00684       long double: "%Lf",                                                      \
00685       default: _Generic((val - val), ptrdiff_t: "%p", default: "undef")),      \
00686       (val))
00687 #else
00688 /*
00689    we don't have the ability to print the values we got, so we create a macro
00690    to tell our users we can't do anything fancy
00691 */
00692 #define utest_type_printer(...) UTEST_PRINTF("undef")
00693 #endif
00694
00695 #if defined(_MSC_VER)
00696 #define UTEST_SURPRESS_WARNING_BEGIN                                           \
00697   __pragma(warning(push)) __pragma(warning(disable : 4127))                    \
00698       __pragma(warning(disable : 4571)) __pragma(warning(disable : 4130))
00699 #define UTEST_SURPRESS_WARNING_END __pragma(warning(pop))
00700 #else
00701 #define UTEST_SURPRESS_WARNING_BEGIN
00702 #define UTEST_SURPRESS_WARNING_END
00703 #endif
00704
00705 #if defined(__cplusplus) && (__cplusplus >= 201103L)
```

```
00706 #define UTEST_AUTO(x) const auto&
00707 #elif !defined(__cplusplus)
00708
00709 #if defined(__clang__)
00710 /* clang-format off */
00711 /* had to disable clang-format here because it malforms the pragmas */
00712 #define UTEST_AUTO(x)                                                       \
00713   _Pragma("clang diagnostic push")                                          \
00714       _Pragma("clang diagnostic ignored \"-Wgnu-auto-type\"") __auto_type   \
00715           _Pragma("clang diagnostic pop")
00716 /* clang-format on */
00717 #else
00718 #define UTEST_AUTO(x) __typeof__(x + 0)
00719 #endif
00720
00721 #else
00722 #define UTEST_AUTO(x) typeof(x + 0)
00723 #endif
00724
00725 #if defined(__clang__)
00726 #define UTEST_STRNCMP(x, y, size)                                           \
00727   _Pragma("clang diagnostic push")                                          \
00728       _Pragma("clang diagnostic ignored \"-Wdisabled-macro-expansion\"")    \
00729           strncmp(x, y, size) _Pragma("clang diagnostic pop")
00730 #else
00731 #define UTEST_STRNCMP(x, y, size) strncmp(x, y, size)
00732 #endif
00733
00734 #if defined(_MSC_VER)
00735 #define UTEST_STRNCPY(x, y, size) strcpy_s(x, size, y)
00736 #elif !defined(__clang__) && defined(__GNUC__)
00737 static UTEST_INLINE char *
00738 utest_strncpy_gcc(char *const dst, const char *const src, const size_t size) {
00739 #pragma GCC diagnostic push
00740 #pragma GCC diagnostic ignored "-Wstringop-overflow"
00741   return strncpy(dst, src, size);
00742 #pragma GCC diagnostic pop
00743 }
00744
00745 #define UTEST_STRNCPY(x, y, size) utest_strncpy_gcc(x, y, size)
00746 #else
00747 #define UTEST_STRNCPY(x, y, size) strncpy(x, y, size)
00748 #endif
00749
00750 #define UTEST_SKIP(msg)                                                     \
00751   do {                                                                      \
00752     UTEST_PRINTF("   Skipped : '%s'\n", (msg));                            \
00753     *utest_result = UTEST_TEST_SKIPPED;                                     \
00754     return;                                                                 \
00755   } while (0)
00756
00757 #if defined(__clang__)
00758 #define UTEST_COND(x, y, cond, msg, is_assert)                              \
00759   UTEST_SURPRESS_WARNING_BEGIN do {                                         \
00760     _Pragma("clang diagnostic push")                                        \
00761         _Pragma("clang diagnostic ignored \"-Wlanguage-extension-token\"")  \
00762             _Pragma("clang diagnostic ignored \"-Wc++98-compat-pedantic\"") \
00763                 _Pragma("clang diagnostic ignored \"-Wfloat-equal\"")       \
00764                     UTEST_AUTO(x) xEval = (x);                              \
00765     UTEST_AUTO(y) yEval = (y);                                              \
00766     if (!((xEval)cond(yEval))) {                                            \
00767       const char *const xAsString = #x;                                     \
00768       const char *const yAsString = #y;                                     \
00769       _Pragma("clang diagnostic pop")                                       \
00770           UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);             \
00771       UTEST_PRINTF("  Expected : (");                                       \
00772       UTEST_PRINTF("%s) " #cond " (%s", xAsString, yAsString);              \
00773       UTEST_PRINTF(")\n");                                                  \
00774       UTEST_PRINTF("    Actual : ");                                        \
00775       utest_type_printer(xEval);                                            \
00776       UTEST_PRINTF(" vs ");                                                 \
00777       utest_type_printer(yEval);                                            \
00778       UTEST_PRINTF("\n");                                                   \
00779       if (strlen(msg) > 0) {                                                \
00780         UTEST_PRINTF("   Message : %s\n", msg);                             \
00781       }                                                                     \
00782       *utest_result = UTEST_TEST_FAILURE;                                   \
00783       if (is_assert) {                                                      \
00784         return;                                                             \
00785       }                                                                     \
00786     }                                                                       \
00787   }                                                                         \
00788   while (0)                                                                 \
00789   UTEST_SURPRESS_WARNING_END
00790 #elif defined(__GNUC__) || defined(__TINYC__)
00791 #define UTEST_COND(x, y, cond, msg, is_assert)                              \
00792   UTEST_SURPRESS_WARNING_BEGIN do {                                         \
```

```
00793     UTEST_AUTO(x) xEval = (x);                                                  \
00794     UTEST_AUTO(y) yEval = (y);                                                  \
00795     if (!((xEval)cond(yEval))) {                                                \
00796       const char *const xAsString = #x;                                         \
00797       const char *const yAsString = #y;                                         \
00798       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                     \
00799       UTEST_PRINTF("  Expected : (");                                           \
00800       UTEST_PRINTF("%s) " #cond " (%s", xAsString, yAsString);                  \
00801       UTEST_PRINTF(")\n");                                                      \
00802       UTEST_PRINTF("    Actual : ");                                            \
00803       utest_type_printer(xEval);                                                \
00804       UTEST_PRINTF(" vs ");                                                     \
00805       utest_type_printer(yEval);                                                \
00806       UTEST_PRINTF("\n");                                                       \
00807       if (strlen(msg) > 0) {                                                    \
00808         UTEST_PRINTF("   Message : %s\n", msg);                               \
00809       }                                                                         \
00810       *utest_result = UTEST_TEST_FAILURE;                                       \
00811       if (is_assert) {                                                          \
00812         return;                                                                 \
00813       }                                                                         \
00814     }                                                                           \
00815   }                                                                             \
00816   while (0)                                                                     \
00817   UTEST_SURPRESS_WARNING_END
00818 #else
00819 #define UTEST_COND(x, y, cond, msg, is_assert)                                  \
00820   UTEST_SURPRESS_WARNING_BEGIN do {                                             \
00821     if (!((x)cond(y))) {                                                        \
00822       UTEST_PRINTF("%s:%i: Failure (Expected " #cond " Actual)", __FILE__,      \
00823                    __LINE__);                                                   \
00824       if (strlen(msg) > 0) {                                                    \
00825         UTEST_PRINTF(" Message : %s", msg);                                     \
00826       }                                                                         \
00827       UTEST_PRINTF("\n");                                                       \
00828       *utest_result = UTEST_TEST_FAILURE;                                       \
00829       if (is_assert) {                                                          \
00830         return;                                                                 \
00831       }                                                                         \
00832     }                                                                           \
00833   }                                                                             \
00834   while (0)                                                                     \
00835   UTEST_SURPRESS_WARNING_END
00836 #endif
00837
00838 #define EXPECT_EQ(x, y) UTEST_COND(x, y, ==, "", 0)
00839 #define EXPECT_EQ_MSG(x, y, msg) UTEST_COND(x, y, ==, msg, 0)
00840 #define ASSERT_EQ(x, y) UTEST_COND(x, y, ==, "", 1)
00841 #define ASSERT_EQ_MSG(x, y, msg) UTEST_COND(x, y, ==, msg, 1)
00842
00843 #define EXPECT_NE(x, y) UTEST_COND(x, y, !=, "", 0)
00844 #define EXPECT_NE_MSG(x, y, msg) UTEST_COND(x, y, !=, msg, 0)
00845 #define ASSERT_NE(x, y) UTEST_COND(x, y, !=, "", 1)
00846 #define ASSERT_NE_MSG(x, y, msg) UTEST_COND(x, y, !=, msg, 1)
00847
00848 #define EXPECT_LT(x, y) UTEST_COND(x, y, <, "", 0)
00849 #define EXPECT_LT_MSG(x, y, msg) UTEST_COND(x, y, <, msg, 0)
00850 #define ASSERT_LT(x, y) UTEST_COND(x, y, <, "", 1)
00851 #define ASSERT_LT_MSG(x, y, msg) UTEST_COND(x, y, <, msg, 1)
00852
00853 #define EXPECT_LE(x, y) UTEST_COND(x, y, <=, "", 0)
00854 #define EXPECT_LE_MSG(x, y, msg) UTEST_COND(x, y, <=, msg, 0)
00855 #define ASSERT_LE(x, y) UTEST_COND(x, y, <=, "", 1)
00856 #define ASSERT_LE_MSG(x, y, msg) UTEST_COND(x, y, <=, msg, 1)
00857
00858 #define EXPECT_GT(x, y) UTEST_COND(x, y, >, "", 0)
00859 #define EXPECT_GT_MSG(x, y, msg) UTEST_COND(x, y, >, msg, 0)
00860 #define ASSERT_GT(x, y) UTEST_COND(x, y, >, "", 1)
00861 #define ASSERT_GT_MSG(x, y, msg) UTEST_COND(x, y, >, msg, 1)
00862
00863 #define EXPECT_GE(x, y) UTEST_COND(x, y, >=, "", 0)
00864 #define EXPECT_GE_MSG(x, y, msg) UTEST_COND(x, y, >=, msg, 0)
00865 #define ASSERT_GE(x, y) UTEST_COND(x, y, >=, "", 1)
00866 #define ASSERT_GE_MSG(x, y, msg) UTEST_COND(x, y, >=, msg, 1)
00867
00868 #define UTEST_TRUE(x, msg, is_assert)                                           \
00869   UTEST_SURPRESS_WARNING_BEGIN do {                                             \
00870     const int xEval = !!(x);                                                    \
00871     if (!(xEval)) {                                                             \
00872       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                     \
00873       UTEST_PRINTF("  Expected : true\n");                                      \
00874       UTEST_PRINTF("    Actual : %s\n", (xEval) ? "true" : "false");            \
00875       if (strlen(msg) > 0) {                                                    \
00876         UTEST_PRINTF("   Message : %s\n", msg);                               \
00877       }                                                                         \
00878       *utest_result = UTEST_TEST_FAILURE;                                       \
00879       if (is_assert) {                                                          \
```

```
00880         return;                                                          \
00881       }                                                                  \
00882     }                                                                    \
00883   }                                                                      \
00884   while (0)                                                              \
00885   UTEST_SURPRESS_WARNING_END
00886
00887 #define EXPECT_TRUE(x) UTEST_TRUE(x, "", 0)
00888 #define EXPECT_TRUE_MSG(x, msg) UTEST_TRUE(x, msg, 0)
00889 #define ASSERT_TRUE(x) UTEST_TRUE(x, "", 1)
00890 #define ASSERT_TRUE_MSG(x, msg) UTEST_TRUE(x, msg, 1)
00891
00892 #define UTEST_FALSE(x, msg, is_assert)                                   \
00893   UTEST_SURPRESS_WARNING_BEGIN do {                                      \
00894     const int xEval = !!(x);                                             \
00895     if (xEval) {                                                         \
00896       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);              \
00897       UTEST_PRINTF("  Expected : false\n");                              \
00898       UTEST_PRINTF("    Actual : %s\n", (xEval) ? "true" : "false");     \
00899       if (strlen(msg) > 0) {                                             \
00900         UTEST_PRINTF("   Message : %s\n", msg);                          \
00901       }                                                                  \
00902       *utest_result = UTEST_TEST_FAILURE;                                \
00903       if (is_assert) {                                                   \
00904         return;                                                          \
00905       }                                                                  \
00906     }                                                                    \
00907   }                                                                      \
00908   while (0)                                                              \
00909   UTEST_SURPRESS_WARNING_END
00910
00911 #define EXPECT_FALSE(x) UTEST_FALSE(x, "", 0)
00912 #define EXPECT_FALSE_MSG(x, msg) UTEST_FALSE(x, msg, 0)
00913 #define ASSERT_FALSE(x) UTEST_FALSE(x, "", 1)
00914 #define ASSERT_FALSE_MSG(x, msg) UTEST_FALSE(x, msg, 1)
00915
00916 #define UTEST_STREQ(x, y, msg, is_assert)                                \
00917   UTEST_SURPRESS_WARNING_BEGIN do {                                      \
00918     const char *xEval = (x);                                             \
00919     const char *yEval = (y);                                             \
00920     if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                    \
00921         0 != strcmp(xEval, yEval)) {                                     \
00922       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);              \
00923       UTEST_PRINTF("  Expected : \"%s\"\n", xEval);                      \
00924       UTEST_PRINTF("    Actual : \"%s\"\n", yEval);                      \
00925       if (strlen(msg) > 0) {                                             \
00926         UTEST_PRINTF("   Message : %s\n", msg);                          \
00927       }                                                                  \
00928       *utest_result = UTEST_TEST_FAILURE;                                \
00929       if (is_assert) {                                                   \
00930         return;                                                          \
00931       }                                                                  \
00932     }                                                                    \
00933   }                                                                      \
00934   while (0)                                                              \
00935   UTEST_SURPRESS_WARNING_END
00936
00937 #define EXPECT_STREQ(x, y) UTEST_STREQ(x, y, "", 0)
00938 #define EXPECT_STREQ_MSG(x, y, msg) UTEST_STREQ(x, y, msg, 0)
00939 #define ASSERT_STREQ(x, y) UTEST_STREQ(x, y, "", 1)
00940 #define ASSERT_STREQ_MSG(x, y, msg) UTEST_STREQ(x, y, msg, 1)
00941
00942 #define UTEST_STRNE(x, y, msg, is_assert)                                \
00943   UTEST_SURPRESS_WARNING_BEGIN do {                                      \
00944     const char *xEval = (x);                                             \
00945     const char *yEval = (y);                                             \
00946     if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                    \
00947         0 == strcmp(xEval, yEval)) {                                     \
00948       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);              \
00949       UTEST_PRINTF("  Expected : \"%s\"\n", xEval);                      \
00950       UTEST_PRINTF("    Actual : \"%s\"\n", yEval);                      \
00951       if (strlen(msg) > 0) {                                             \
00952         UTEST_PRINTF("   Message : %s\n", msg);                          \
00953       }                                                                  \
00954       *utest_result = UTEST_TEST_FAILURE;                                \
00955       if (is_assert) {                                                   \
00956         return;                                                          \
00957       }                                                                  \
00958     }                                                                    \
00959   }                                                                      \
00960   while (0)                                                              \
00961   UTEST_SURPRESS_WARNING_END
00962
00963 #define EXPECT_STRNE(x, y) UTEST_STRNE(x, y, "", 0)
00964 #define EXPECT_STRNE_MSG(x, y, msg) UTEST_STRNE(x, y, msg, 0)
00965 #define ASSERT_STRNE(x, y) UTEST_STRNE(x, y, "", 1)
00966 #define ASSERT_STRNE_MSG(x, y, msg) UTEST_STRNE(x, y, msg, 1)
```

```
00967
00968 #define UTEST_STRNEQ(x, y, n, msg, is_assert)                                   \
00969   UTEST_SURPRESS_WARNING_BEGIN do {                                             \
00970     const char *xEval = (x);                                                    \
00971     const char *yEval = (y);                                                    \
00972     const size_t nEval = UTEST_CAST(size_t, n);                                 \
00973     if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                           \
00974         0 != UTEST_STRNCMP(xEval, yEval, nEval)) {                              \
00975       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                     \
00976       UTEST_PRINTF("  Expected : \"%.*s\"\n", UTEST_CAST(int, nEval), xEval);   \
00977       UTEST_PRINTF("    Actual : \"%.*s\"\n", UTEST_CAST(int, nEval), yEval);   \
00978       if (strlen(msg) > 0) {                                                    \
00979         UTEST_PRINTF("   Message : %s\n", msg);                               \
00980       }                                                                         \
00981       *utest_result = UTEST_TEST_FAILURE;                                       \
00982       if (is_assert) {                                                          \
00983         return;                                                                 \
00984       }                                                                         \
00985     }                                                                           \
00986   }                                                                             \
00987   while (0)                                                                     \
00988   UTEST_SURPRESS_WARNING_END
00989
00990 #define EXPECT_STRNEQ(x, y, n) UTEST_STRNEQ(x, y, n, "", 0)
00991 #define EXPECT_STRNEQ_MSG(x, y, n, msg) UTEST_STRNEQ(x, y, n, msg, 0)
00992 #define ASSERT_STRNEQ(x, y, n) UTEST_STRNEQ(x, y, n, "", 1)
00993 #define ASSERT_STRNEQ_MSG(x, y, n, msg) UTEST_STRNEQ(x, y, n, msg, 1)
00994
00995 #define UTEST_STRNNE(x, y, n, msg, is_assert)                                   \
00996   UTEST_SURPRESS_WARNING_BEGIN do {                                             \
00997     const char *xEval = (x);                                                    \
00998     const char *yEval = (y);                                                    \
00999     const size_t nEval = UTEST_CAST(size_t, n);                                 \
01000     if (UTEST_NULL == xEval || UTEST_NULL == yEval ||                           \
01001         0 == UTEST_STRNCMP(xEval, yEval, nEval)) {                              \
01002       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                     \
01003       UTEST_PRINTF("  Expected : \"%.*s\"\n", UTEST_CAST(int, nEval), xEval);   \
01004       UTEST_PRINTF("    Actual : \"%.*s\"\n", UTEST_CAST(int, nEval), yEval);   \
01005       if (strlen(msg) > 0) {                                                    \
01006         UTEST_PRINTF("   Message : %s\n", msg);                               \
01007       }                                                                         \
01008       *utest_result = UTEST_TEST_FAILURE;                                       \
01009       if (is_assert) {                                                          \
01010         return;                                                                 \
01011       }                                                                         \
01012     }                                                                           \
01013   }                                                                             \
01014   while (0)                                                                     \
01015   UTEST_SURPRESS_WARNING_END
01016
01017 #define EXPECT_STRNNE(x, y, n) UTEST_STRNNE(x, y, n, "", 0)
01018 #define EXPECT_STRNNE_MSG(x, y, n, msg) UTEST_STRNNE(x, y, n, msg, 0)
01019 #define ASSERT_STRNNE(x, y, n) UTEST_STRNNE(x, y, n, "", 1)
01020 #define ASSERT_STRNNE_MSG(x, y, n, msg) UTEST_STRNNE(x, y, n, msg, 1)
01021
01022 #define UTEST_NEAR(x, y, epsilon, msg, is_assert)                               \
01023   UTEST_SURPRESS_WARNING_BEGIN do {                                             \
01024     const double diff =                                                         \
01025         utest_fabs(UTEST_CAST(double, x) - UTEST_CAST(double, y));              \
01026     if (diff > UTEST_CAST(double, epsilon) || utest_isnan(diff)) {              \
01027       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                     \
01028       UTEST_PRINTF("  Expected : %f\n", UTEST_CAST(double, x));                 \
01029       UTEST_PRINTF("    Actual : %f\n", UTEST_CAST(double, y));                 \
01030       if (strlen(msg) > 0) {                                                    \
01031         UTEST_PRINTF("   Message : %s\n", msg);                               \
01032       }                                                                         \
01033       *utest_result = UTEST_TEST_FAILURE;                                       \
01034       if (is_assert) {                                                          \
01035         return;                                                                 \
01036       }                                                                         \
01037     }                                                                           \
01038   }                                                                             \
01039   while (0)                                                                     \
01040   UTEST_SURPRESS_WARNING_END
01041
01042 #define EXPECT_NEAR(x, y, epsilon) UTEST_NEAR(x, y, epsilon, "", 0)
01043 #define EXPECT_NEAR_MSG(x, y, epsilon, msg) UTEST_NEAR(x, y, epsilon, msg, 0)
01044 #define ASSERT_NEAR(x, y, epsilon) UTEST_NEAR(x, y, epsilon, "", 1)
01045 #define ASSERT_NEAR_MSG(x, y, epsilon, msg) UTEST_NEAR(x, y, epsilon, msg, 1)
01046
01047 #if defined(UTEST_HAS_EXCEPTIONS)
01048 #define UTEST_EXCEPTION(x, exception_type, msg, is_assert)                      \
01049   UTEST_SURPRESS_WARNING_BEGIN do {                                             \
01050     int exception_caught = 0;                                                   \
01051     try {                                                                       \
01052       x;                                                                        \
01053     } catch (const exception_type &) {                                          \
```

```
01054       exception_caught = 1;                                                \
01055     } catch (...) {                                                        \
01056       exception_caught = 2;                                                \
01057     }                                                                      \
01058     if (1 != exception_caught) {                                           \
01059       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                \
01060       UTEST_PRINTF("  Expected : %s exception\n", #exception_type);         \
01061       UTEST_PRINTF("    Actual : %s\n", (2 == exception_caught)            \
01062                                           ? "Unexpected exception"         \
01063                                           : "No exception");               \
01064       if (strlen(msg) > 0) {                                               \
01065         UTEST_PRINTF("   Message : %s\n", msg);                            \
01066       }                                                                    \
01067       *utest_result = UTEST_TEST_FAILURE;                                  \
01068       if (is_assert) {                                                     \
01069         return;                                                            \
01070       }                                                                    \
01071     }                                                                      \
01072   }                                                                        \
01073   while (0)                                                                \
01074   UTEST_SURPRESS_WARNING_END
01075
01076 #define EXPECT_EXCEPTION(x, exception_type)                                \
01077   UTEST_EXCEPTION(x, exception_type, "", 0)
01078 #define EXPECT_EXCEPTION_MSG(x, exception_type, msg)                       \
01079   UTEST_EXCEPTION(x, exception_type, msg, 0)
01080 #define ASSERT_EXCEPTION(x, exception_type)                                \
01081   UTEST_EXCEPTION(x, exception_type, "", 1)
01082 #define ASSERT_EXCEPTION_MSG(x, exception_type, msg)                       \
01083   UTEST_EXCEPTION(x, exception_type, msg, 1)
01084
01085 #define UTEST_EXCEPTION_WITH_MESSAGE(x, exception_type, exception_message,     \
01086                                      msg, is_assert)                       \
01087   UTEST_SURPRESS_WARNING_BEGIN do {                                        \
01088     int exception_caught = 0;                                              \
01089     char *message_caught = UTEST_NULL;                                     \
01090     try {                                                                  \
01091       x;                                                                   \
01092     } catch (const exception_type &e) {                                    \
01093       const char *const what = e.what();                                   \
01094       exception_caught = 1;                                                \
01095       if (0 !=                                                             \
01096           UTEST_STRNCMP(what, exception_message, strlen(exception_message))) { \
01097         const size_t message_size = strlen(what) + 1;                      \
01098         message_caught = UTEST_PTR_CAST(char *, malloc(message_size));     \
01099         UTEST_STRNCPY(message_caught, what, message_size);                 \
01100       }                                                                    \
01101     } catch (...) {                                                        \
01102       exception_caught = 2;                                                \
01103     }                                                                      \
01104     if (1 != exception_caught) {                                           \
01105       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                \
01106       UTEST_PRINTF("  Expected : %s exception\n", #exception_type);         \
01107       UTEST_PRINTF("    Actual : %s\n", (2 == exception_caught)            \
01108                                           ? "Unexpected exception"         \
01109                                           : "No exception");               \
01110       if (strlen(msg) > 0) {                                               \
01111         UTEST_PRINTF("   Message : %s\n", msg);                            \
01112       }                                                                    \
01113       *utest_result = UTEST_TEST_FAILURE;                                  \
01114       if (is_assert) {                                                     \
01115         return;                                                            \
01116       }                                                                    \
01117     } else if (UTEST_NULL != message_caught) {                             \
01118       UTEST_PRINTF("%s:%i: Failure\n", __FILE__, __LINE__);                \
01119       UTEST_PRINTF("  Expected : %s exception with message %s\n",          \
01120                    #exception_type, exception_message);                    \
01121       UTEST_PRINTF("    Actual message : %s\n", message_caught);           \
01122       if (strlen(msg) > 0) {                                               \
01123         UTEST_PRINTF("   Message : %s\n", msg);                            \
01124       }                                                                    \
01125       *utest_result = UTEST_TEST_FAILURE;                                  \
01126       free(message_caught);                                                \
01127       if (is_assert) {                                                     \
01128         return;                                                            \
01129       }                                                                    \
01130     }                                                                      \
01131   }                                                                        \
01132   while (0)                                                                \
01133   UTEST_SURPRESS_WARNING_END
01134
01135 #define EXPECT_EXCEPTION_WITH_MESSAGE(x, exception_type, exception_message)    \
01136   UTEST_EXCEPTION_WITH_MESSAGE(x, exception_type, exception_message, "", 0)
01137 #define EXPECT_EXCEPTION_WITH_MESSAGE_MSG(x, exception_type,                   \
01138                                           exception_message, msg)          \
01139   UTEST_EXCEPTION_WITH_MESSAGE(x, exception_type, exception_message, msg, 0)
01140 #define ASSERT_EXCEPTION_WITH_MESSAGE(x, exception_type, exception_message)    \
```

```
01141   UTEST_EXCEPTION_WITH_MESSAGE(x, exception_type, exception_message, "", 1)
01142 #define ASSERT_EXCEPTION_WITH_MESSAGE_MSG(x, exception_type,                    \
01143                                           exception_message, msg)              \
01144   UTEST_EXCEPTION_WITH_MESSAGE(x, exception_type, exception_message, msg, 1)
01145 #endif
01146
01147 #if defined(__clang__)
01148 #if __has_warning("-Wunsafe-buffer-usage")
01149 #define UTEST_SURPRESS_WARNINGS_BEGIN                                          \
01150   _Pragma("clang diagnostic push")                                             \
01151       _Pragma("clang diagnostic ignored \"-Wunsafe-buffer-usage\"")
01152 #define UTEST_SURPRESS_WARNINGS_END _Pragma("clang diagnostic pop")
01153 #else
01154 #define UTEST_SURPRESS_WARNINGS_BEGIN
01155 #define UTEST_SURPRESS_WARNINGS_END
01156 #endif
01157 #elif defined(__GNUC__) && __GNUC__ >= 8 && defined(__cplusplus)
01158 #define UTEST_SURPRESS_WARNINGS_BEGIN                                          \
01159   _Pragma("GCC diagnostic push")                                               \
01160       _Pragma("GCC diagnostic ignored \"-Wclass-memaccess\"")
01161 #define UTEST_SURPRESS_WARNINGS_END _Pragma("GCC diagnostic pop")
01162 #else
01163 #define UTEST_SURPRESS_WARNINGS_BEGIN
01164 #define UTEST_SURPRESS_WARNINGS_END
01165 #endif
01166
01167 #define UTEST(SET, NAME)                                                       \
01168   UTEST_SURPRESS_WARNINGS_BEGIN                                                \
01169   UTEST_EXTERN struct utest_state_s utest_state;                               \
01170   static void utest_run_##SET##_##NAME(int *utest_result);                     \
01171   static void utest_##SET##_##NAME(int *utest_result, size_t utest_index) {    \
01172     (void)utest_index;                                                         \
01173     utest_run_##SET##_##NAME(utest_result);                                    \
01174   }                                                                            \
01175   UTEST_INITIALIZER(utest_register_##SET##_##NAME) {                           \
01176     const size_t index = utest_state.tests_length++;                           \
01177     const char name_part[] = #SET "." #NAME;                                   \
01178     const size_t name_size = strlen(name_part) + 1;                            \
01179     char *name = UTEST_PTR_CAST(char *, malloc(name_size));                    \
01180     utest_state.tests = UTEST_PTR_CAST(                                        \
01181         struct utest_test_state_s *,                                           \
01182         utest_realloc(UTEST_PTR_CAST(void *, utest_state.tests),               \
01183                       sizeof(struct utest_test_state_s) *                      \
01184                           utest_state.tests_length));                          \
01185     if (utest_state.tests && name) {                                           \
01186       utest_state.tests[index].func = &utest_##SET##_##NAME;                   \
01187       utest_state.tests[index].name = name;                                    \
01188       utest_state.tests[index].index = 0;                                      \
01189       UTEST_SNPRINTF(name, name_size, "%s", name_part);                        \
01190     } else {                                                                   \
01191       if (utest_state.tests) {                                                 \
01192         free(utest_state.tests);                                               \
01193         utest_state.tests = NULL;                                              \
01194       }                                                                        \
01195       if (name) {                                                              \
01196         free(name);                                                            \
01197       }                                                                        \
01198     }                                                                          \
01199   }                                                                            \
01200   UTEST_SURPRESS_WARNINGS_END                                                  \
01201   void utest_run_##SET##_##NAME(int *utest_result)
01202
01203 #define UTEST_F_SETUP(FIXTURE)                                                 \
01204   static void utest_f_setup_##FIXTURE(int *utest_result,                       \
01205                                       struct FIXTURE *utest_fixture)
01206
01207 #define UTEST_F_TEARDOWN(FIXTURE)                                              \
01208   static void utest_f_teardown_##FIXTURE(int *utest_result,                    \
01209                                          struct FIXTURE *utest_fixture)
01210
01211 #define UTEST_F(FIXTURE, NAME)                                                 \
01212   UTEST_SURPRESS_WARNINGS_BEGIN                                                \
01213   UTEST_EXTERN struct utest_state_s utest_state;                               \
01214   static void utest_f_setup_##FIXTURE(int *, struct FIXTURE *);                \
01215   static void utest_f_teardown_##FIXTURE(int *, struct FIXTURE *);             \
01216   static void utest_run_##FIXTURE##_##NAME(int *, struct FIXTURE *);           \
01217   static void utest_f_##FIXTURE##_##NAME(int *utest_result,                    \
01218                                          size_t utest_index) {                 \
01219     struct FIXTURE fixture;                                                    \
01220     (void)utest_index;                                                         \
01221     memset(&fixture, 0, sizeof(fixture));                                      \
01222     utest_f_setup_##FIXTURE(utest_result, &fixture);                           \
01223     if (UTEST_TEST_PASSED != *utest_result) {                                  \
01224       return;                                                                  \
01225     }                                                                          \
01226     utest_run_##FIXTURE##_##NAME(utest_result, &fixture);                      \
01227     utest_f_teardown_##FIXTURE(utest_result, &fixture);                        \
```

```
01228   }                                                                       \
01229   UTEST_INITIALIZER(utest_register_##FIXTURE##_##NAME) {                  \
01230     const size_t index = utest_state.tests_length++;                      \
01231     const char name_part[] = #FIXTURE "." #NAME;                          \
01232     const size_t name_size = strlen(name_part) + 1;                       \
01233     char *name = UTEST_PTR_CAST(char *, malloc(name_size));               \
01234     utest_state.tests = UTEST_PTR_CAST(                                   \
01235         struct utest_test_state_s *,                                      \
01236         utest_realloc(UTEST_PTR_CAST(void *, utest_state.tests),          \
01237                       sizeof(struct utest_test_state_s) *                 \
01238                           utest_state.tests_length));                     \
01239     if (utest_state.tests && name) {                                      \
01240       utest_state.tests[index].func = &utest_f_##FIXTURE##_##NAME;        \
01241       utest_state.tests[index].name = name;                               \
01242       utest_state.tests[index].index = 0;                                 \
01243       UTEST_SNPRINTF(name, name_size, "%s", name_part);                   \
01244     } else {                                                              \
01245       if (utest_state.tests) {                                            \
01246         free(utest_state.tests);                                          \
01247         utest_state.tests = NULL;                                         \
01248       }                                                                   \
01249       if (name) {                                                         \
01250         free(name);                                                       \
01251       }                                                                   \
01252     }                                                                     \
01253   }                                                                       \
01254   UTEST_SURPRESS_WARNINGS_END                                             \
01255   void utest_run_##FIXTURE##_##NAME(int *utest_result,                    \
01256                                     struct FIXTURE *utest_fixture)
01257
01258 #define UTEST_I_SETUP(FIXTURE)                                            \
01259   static void utest_i_setup_##FIXTURE(                                    \
01260       int *utest_result, struct FIXTURE *utest_fixture, size_t utest_index)
01261
01262 #define UTEST_I_TEARDOWN(FIXTURE)                                         \
01263   static void utest_i_teardown_##FIXTURE(                                 \
01264       int *utest_result, struct FIXTURE *utest_fixture, size_t utest_index)
01265
01266 #define UTEST_I(FIXTURE, NAME, INDEX)                                     \
01267   UTEST_SURPRESS_WARNINGS_BEGIN                                           \
01268   UTEST_EXTERN struct utest_state_s utest_state;                          \
01269   static void utest_run_##FIXTURE##_##NAME##_##INDEX(int *, struct FIXTURE *); \
01270   static void utest_i_##FIXTURE##_##NAME##_##INDEX(int *utest_result,     \
01271                                                    size_t index) {        \
01272     struct FIXTURE fixture;                                               \
01273     memset(&fixture, 0, sizeof(fixture));                                 \
01274     utest_i_setup_##FIXTURE(utest_result, &fixture, index);               \
01275     if (UTEST_TEST_PASSED != *utest_result) {                             \
01276       return;                                                             \
01277     }                                                                     \
01278     utest_run_##FIXTURE##_##NAME##_##INDEX(utest_result, &fixture);       \
01279     utest_i_teardown_##FIXTURE(utest_result, &fixture, index);            \
01280   }                                                                       \
01281   UTEST_INITIALIZER(utest_register_##FIXTURE##_##NAME##_##INDEX) {        \
01282     size_t i;                                                             \
01283     utest_uint64_t iUp;                                                   \
01284     for (i = 0; i < (INDEX); i++) {                                       \
01285       const size_t index = utest_state.tests_length++;                    \
01286       const char name_part[] = #FIXTURE "." #NAME;                        \
01287       const size_t name_size = strlen(name_part) + 32;                    \
01288       char *name = UTEST_PTR_CAST(char *, malloc(name_size));             \
01289       utest_state.tests = UTEST_PTR_CAST(                                 \
01290           struct utest_test_state_s *,                                    \
01291           utest_realloc(UTEST_PTR_CAST(void *, utest_state.tests),        \
01292                         sizeof(struct utest_test_state_s) *               \
01293                             utest_state.tests_length));                   \
01294       if (utest_state.tests && name) {                                    \
01295         utest_state.tests[index].func = &utest_i_##FIXTURE##_##NAME##_##INDEX; \
01296         utest_state.tests[index].index = i;                               \
01297         utest_state.tests[index].name = name;                             \
01298         iUp = UTEST_CAST(utest_uint64_t, i);                              \
01299         UTEST_SNPRINTF(name, name_size, "%s/%" UTEST_PRIu64, name_part, iUp); \
01300       } else {                                                            \
01301         if (utest_state.tests) {                                          \
01302           free(utest_state.tests);                                        \
01303           utest_state.tests = NULL;                                       \
01304         }                                                                 \
01305         if (name) {                                                       \
01306           free(name);                                                     \
01307         }                                                                 \
01308       }                                                                   \
01309     }                                                                     \
01310   }                                                                       \
01311   UTEST_SURPRESS_WARNINGS_END                                             \
01312   void utest_run_##FIXTURE##_##NAME##_##INDEX(int *utest_result,          \
01313                                               struct FIXTURE *utest_fixture)
01314
```

```
01315 #ifdef __clang__
01316 #pragma clang diagnostic push
01317 #pragma clang diagnostic ignored "-Wc++98-compat-pedantic"
01318 #endif
01319
01320 UTEST_WEAK
01321 double utest_fabs(double d);
01322 UTEST_WEAK
01323 double utest_fabs(double d) {
01324   union {
01325     double d;
01326     utest_uint64_t u;
01327   } both;
01328   both.d = d;
01329   both.u &= 0x7fffffffffffffffu;
01330   return both.d;
01331 }
01332
01333 UTEST_WEAK
01334 int utest_isnan(double d);
01335 UTEST_WEAK
01336 int utest_isnan(double d) {
01337   union {
01338     double d;
01339     utest_uint64_t u;
01340   } both;
01341   both.d = d;
01342   both.u &= 0x7fffffffffffffffu;
01343   return both.u > 0x7ff0000000000000u;
01344 }
01345
01346 #ifdef __clang__
01347 #pragma clang diagnostic pop
01348 #endif
01349
01350 #if defined(__clang__)
01351 #if __has_warning("-Wunsafe-buffer-usage")
01352 #pragma clang diagnostic push
01353 #pragma clang diagnostic ignored "-Wunsafe-buffer-usage"
01354 #endif
01355 #endif
01356
01357 UTEST_WEAK
01358 int utest_should_filter_test(const char *filter, const char *testcase);
01359 UTEST_WEAK int utest_should_filter_test(const char *filter,
01360                                         const char *testcase) {
01361   if (filter) {
01362     const char *filter_cur = filter;
01363     const char *testcase_cur = testcase;
01364     const char *filter_wildcard = UTEST_NULL;
01365
01366     while (('\0' != *filter_cur) && ('\0' != *testcase_cur)) {
01367       if ('*' == *filter_cur) {
01368         /* store the position of the wildcard */
01369         filter_wildcard = filter_cur;
01370
01371         /* skip the wildcard character */
01372         filter_cur++;
01373
01374         while (('\0' != *filter_cur) && ('\0' != *testcase_cur)) {
01375           if ('*' == *filter_cur) {
01376             /*
01377                we found another wildcard (filter is something like *foo*) so we
01378                exit the current loop, and return to the parent loop to handle
01379                the wildcard case
01380             */
01381             break;
01382           } else if (*filter_cur != *testcase_cur) {
01383             /* otherwise our filter didn't match, so reset it */
01384             filter_cur = filter_wildcard;
01385           }
01386
01387           /* move testcase along */
01388           testcase_cur++;
01389
01390           /* move filter along */
01391           filter_cur++;
01392         }
01393
01394         if (('\0' == *filter_cur) && ('\0' == *testcase_cur)) {
01395           return 0;
01396         }
01397
01398         /* if the testcase has been exhausted, we don't have a match! */
01399         if ('\0' == *testcase_cur) {
01400           return 1;
01401         }
```

```
01402       } else {
01403         if (*testcase_cur != *filter_cur) {
01404           /* test case doesn't match filter */
01405           return 1;
01406         } else {
01407           /* move our filter and testcase forward */
01408           testcase_cur++;
01409           filter_cur++;
01410         }
01411       }
01412     }
01413
01414     if (('\0' != *filter_cur) ||
01415         ('\0' != *testcase_cur) &&
01416          ((filter == filter_cur) || ('*' != filter_cur[-1])))) {
01417       /* we have a mismatch! */
01418       return 1;
01419     }
01420   }
01421
01422   return 0;
01423 }
01424
01425 static UTEST_INLINE FILE *utest_fopen(const char *filename, const char *mode) {
01426 #ifdef _MSC_VER
01427   FILE *file;
01428   if (0 == fopen_s(&file, filename, mode)) {
01429     return file;
01430   } else {
01431     return UTEST_NULL;
01432   }
01433 #else
01434   return fopen(filename, mode);
01435 #endif
01436 }
01437
01438 static UTEST_INLINE int utest_main(int argc, const char *const argv[]);
01439 int utest_main(int argc, const char *const argv[]) {
01440   utest_uint64_t failed = 0;
01441   utest_uint64_t skipped = 0;
01442   size_t index = 0;
01443   size_t *failed_testcases = UTEST_NULL;
01444   size_t failed_testcases_length = 0;
01445   size_t *skipped_testcases = UTEST_NULL;
01446   size_t skipped_testcases_length = 0;
01447   const char *filter = UTEST_NULL;
01448   utest_uint64_t ran_tests = 0;
01449   int enable_mixed_units = 0;
01450   int random_order = 0;
01451   utest_uint32_t seed = 0;
01452
01453   enum colours { RESET, GREEN, RED, YELLOW };
01454
01455   const int use_colours = UTEST_COLOUR_OUTPUT();
01456   const char *colours[] = {"\033[0m", "\033[32m", "\033[31m", "\033[33m"};
01457
01458   if (!use_colours) {
01459     for (index = 0; index < sizeof colours / sizeof colours[0]; index++) {
01460       colours[index] = "";
01461     }
01462   }
01463   /* loop through all arguments looking for our options */
01464   for (index = 1; index < UTEST_CAST(size_t, argc); index++) {
01465     /* Informational switches */
01466     const char help_str[] = "--help";
01467     const char list_str[] = "--list-tests";
01468     /* Test config switches */
01469     const char filter_str[] = "--filter=";
01470     const char output_str[] = "--output=";
01471     const char enable_mixed_units_str[] = "--enable-mixed-units";
01472     const char random_order_str[] = "--random-order";
01473     const char random_order_with_seed_str[] = "--random-order=";
01474
01475     if (0 == UTEST_STRNCMP(argv[index], help_str, strlen(help_str))) {
01476       printf("utest.h - the single file unit testing solution for C/C++!\n"
01477              "Command line Options:\n"
01478              "  --help                 Show this message and exit.\n"
01479              "  --filter=<filter>      Filter the test cases to run (EG. "
01480              "MyTest*.a would run MyTestCase.a but not MyTestCase.b).\n"
01481              "  --list-tests           List testnames, one per line. Output "
01482              "names can be passed to --filter.\n");
01483       printf("  --output=<output>      Output an xunit XML file to the file "
01484              "specified in <output>.\n"
01485              "  --enable-mixed-units   Enable the per-test output to contain "
01486              "mixed units (s/ms/us/ns).\n"
01487              "  --random-order[=<seed>] Randomize the order that the tests are "
01488              "ran in. If the optional <seed> argument is not provided, then a "
```

```
01489             "random starting seed is used.\n");
01490       goto cleanup;
01491     } else if (0 ==
01492                UTEST_STRNCMP(argv[index], filter_str, strlen(filter_str))) {
01493       /* user wants to filter what test cases run! */
01494       filter = argv[index] + strlen(filter_str);
01495     } else if (0 ==
01496                UTEST_STRNCMP(argv[index], output_str, strlen(output_str))) {
01497       utest_state.output = utest_fopen(argv[index] + strlen(output_str), "w+");
01498     } else if (0 == UTEST_STRNCMP(argv[index], list_str, strlen(list_str))) {
01499       for (index = 0; index < utest_state.tests_length; index++) {
01500         UTEST_PRINTF("%s\n", utest_state.tests[index].name);
01501       }
01502       /* when printing the test list, don't actually run the tests */
01503       return 0;
01504     } else if (0 == UTEST_STRNCMP(argv[index], enable_mixed_units_str,
01505                                   strlen(enable_mixed_units_str))) {
01506       enable_mixed_units = 1;
01507     } else if (0 == UTEST_STRNCMP(argv[index], random_order_with_seed_str,
01508                                   strlen(random_order_with_seed_str))) {
01509       seed =
01510           UTEST_CAST(utest_uint32_t,
01511                      strtoul(argv[index] + strlen(random_order_with_seed_str),
01512                              UTEST_NULL, 10));
01513       random_order = 1;
01514     } else if (0 == UTEST_STRNCMP(argv[index], random_order_str,
01515                                   strlen(random_order_str))) {
01516       const utest_int64_t ns = utest_ns();
01517
01518       // Some really poor pseudo-random using the current time. I do this
01519       // because I really want to avoid using C's rand() because that'd mean our
01520       // random would be affected by any srand() usage by the user (which I
01521       // don't want).
01522       seed = UTEST_CAST(utest_uint32_t, ns >> 32) * 31 +
01523              UTEST_CAST(utest_uint32_t, ns & 0xffffffff);
01524       random_order = 1;
01525     }
01526   }
01527
01528   if (random_order) {
01529     // Use Fisher-Yates with the Durstenfield's version to randomly re-order the
01530     // tests.
01531     for (index = utest_state.tests_length; index > 1; index--) {
01532       // For the random order we'll use PCG.
01533       const utest_uint32_t state = seed;
01534       const utest_uint32_t word =
01535           ((state >> ((state >> 28u) + 4u)) ^ state) * 277803737u;
01536       const utest_uint32_t next =
01537           ((word >> 22u) ^ word) % UTEST_CAST(utest_uint32_t, index);
01538
01539       // Swap the randomly chosen element into the last location.
01540       const struct utest_test_state_s copy = utest_state.tests[index - 1];
01541       utest_state.tests[index - 1] = utest_state.tests[next];
01542       utest_state.tests[next] = copy;
01543
01544       // Move the seed onwards.
01545       seed = seed * 747796405u + 2891336453u;
01546     }
01547   }
01548
01549   for (index = 0; index < utest_state.tests_length; index++) {
01550     if (utest_should_filter_test(filter, utest_state.tests[index].name)) {
01551       continue;
01552     }
01553
01554     ran_tests++;
01555   }
01556
01557   printf("%s[==========]%s Running %" UTEST_PRIu64 " test cases.\n",
01558          colours[GREEN], colours[RESET], UTEST_CAST(utest_uint64_t, ran_tests));
01559
01560   if (utest_state.output) {
01561     fprintf(utest_state.output, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
01562     fprintf(utest_state.output,
01563             "<testsuites tests=\"%" UTEST_PRIu64 "\" name=\"All\">\n",
01564             UTEST_CAST(utest_uint64_t, ran_tests));
01565     fprintf(utest_state.output,
01566             "<testsuite name=\"Tests\" tests=\"%" UTEST_PRIu64 "\">\n",
01567             UTEST_CAST(utest_uint64_t, ran_tests));
01568   }
01569
01570   for (index = 0; index < utest_state.tests_length; index++) {
01571     int result = UTEST_TEST_PASSED;
01572     utest_int64_t ns = 0;
01573
01574     if (utest_should_filter_test(filter, utest_state.tests[index].name)) {
01575       continue;
```

```
01576     }
01577
01578     printf("%s[ RUN      ]%s %s\n", colours[GREEN], colours[RESET],
01579            utest_state.tests[index].name);
01580
01581     if (utest_state.output) {
01582       fprintf(utest_state.output, "<testcase name=\"%s\">",
01583               utest_state.tests[index].name);
01584     }
01585
01586     ns = utest_ns();
01587     errno = 0;
01588 #if defined(UTEST_HAS_EXCEPTIONS)
01589     UTEST_SURPRESS_WARNING_BEGIN
01590     try {
01591       utest_state.tests[index].func(&result, utest_state.tests[index].index);
01592     } catch (const std::exception &err) {
01593       printf(" Exception : %s\n", err.what());
01594       result = UTEST_TEST_FAILURE;
01595     } catch (...) {
01596       printf(" Exception : Unknown\n");
01597       result = UTEST_TEST_FAILURE;
01598     }
01599     UTEST_SURPRESS_WARNING_END
01600 #else
01601     utest_state.tests[index].func(&result, utest_state.tests[index].index);
01602 #endif
01603     ns = utest_ns() - ns;
01604
01605     if (utest_state.output) {
01606       fprintf(utest_state.output, "</testcase>\n");
01607     }
01608
01609     // Record the failing test.
01610     if (UTEST_TEST_FAILURE == result) {
01611       const size_t failed_testcase_index = failed_testcases_length++;
01612       failed_testcases = UTEST_PTR_CAST(
01613           size_t *, utest_realloc(UTEST_PTR_CAST(void *, failed_testcases),
01614                                   sizeof(size_t) * failed_testcases_length));
01615       if (UTEST_NULL != failed_testcases) {
01616         failed_testcases[failed_testcase_index] = index;
01617       }
01618       failed++;
01619     } else if (UTEST_TEST_SKIPPED == result) {
01620       const size_t skipped_testcase_index = skipped_testcases_length++;
01621       skipped_testcases = UTEST_PTR_CAST(
01622           size_t *, utest_realloc(UTEST_PTR_CAST(void *, skipped_testcases),
01623                                   sizeof(size_t) * skipped_testcases_length));
01624       if (UTEST_NULL != skipped_testcases) {
01625         skipped_testcases[skipped_testcase_index] = index;
01626       }
01627       skipped++;
01628     }
01629
01630     {
01631       const char *const units[] = {"ns", "us", "ms", "s", UTEST_NULL};
01632       unsigned int unit_index = 0;
01633       utest_int64_t time = ns;
01634
01635       if (enable_mixed_units) {
01636         for (unit_index = 0; UTEST_NULL != units[unit_index]; unit_index++) {
01637           if (10000 > time) {
01638             break;
01639           }
01640
01641           time /= 1000;
01642         }
01643       }
01644
01645       if (UTEST_TEST_FAILURE == result) {
01646         printf("%s[  FAILED  ]%s %s (%" UTEST_PRId64 "%s)\n", colours[RED],
01647                colours[RESET], utest_state.tests[index].name, time,
01648                units[unit_index]);
01649       } else if (UTEST_TEST_SKIPPED == result) {
01650         printf("%s[  SKIPPED ]%s %s (%" UTEST_PRId64 "%s)\n", colours[YELLOW],
01651                colours[RESET], utest_state.tests[index].name, time,
01652                units[unit_index]);
01653       } else {
01654         printf("%s[       OK ]%s %s (%" UTEST_PRId64 "%s)\n", colours[GREEN],
01655                colours[RESET], utest_state.tests[index].name, time,
01656                units[unit_index]);
01657       }
01658     }
01659   }
01660
01661   printf("%s[==========]%s %" UTEST_PRIu64 " test cases ran.\n", colours[GREEN],
01662          colours[RESET], ran_tests);
```

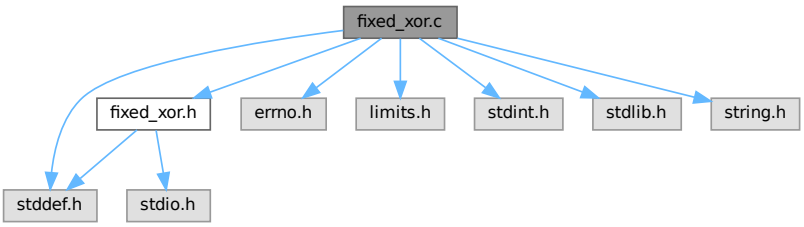
```
01663   printf("%s[  PASSED  ]%s %" UTEST_PRIu64 " tests.\n", colours[GREEN],
01664          colours[RESET], ran_tests - failed - skipped);
01665
01666   if (0 != skipped) {
01667     printf("%s[  SKIPPED ]%s %" UTEST_PRIu64 " tests, listed below:\n",
01668            colours[YELLOW], colours[RESET], skipped);
01669     for (index = 0; index < skipped_testcases_length; index++) {
01670       printf("%s[  SKIPPED ]%s %s\n", colours[YELLOW], colours[RESET],
01671              utest_state.tests[skipped_testcases[index]].name);
01672     }
01673   }
01674
01675   if (0 != failed) {
01676     printf("%s[  FAILED  ]%s %" UTEST_PRIu64 " tests, listed below:\n",
01677            colours[RED], colours[RESET], failed);
01678     for (index = 0; index < failed_testcases_length; index++) {
01679       printf("%s[  FAILED  ]%s %s\n", colours[RED], colours[RESET],
01680              utest_state.tests[failed_testcases[index]].name);
01681     }
01682   }
01683
01684   if (utest_state.output) {
01685     fprintf(utest_state.output, "</testsuite>\n</testsuites>\n");
01686   }
01687
01688 cleanup:
01689   for (index = 0; index < utest_state.tests_length; index++) {
01690     free(UTEST_PTR_CAST(void *, utest_state.tests[index].name));
01691   }
01692
01693   free(UTEST_PTR_CAST(void *, skipped_testcases));
01694   free(UTEST_PTR_CAST(void *, failed_testcases));
01695   free(UTEST_PTR_CAST(void *, utest_state.tests));
01696
01697   if (utest_state.output) {
01698     fclose(utest_state.output);
01699   }
01700
01701   return UTEST_CAST(int, failed);
01702 }
01703
01704 #if defined(__clang__)
01705 #if __has_warning("-Wunsafe-buffer-usage")
01706 #pragma clang diagnostic pop
01707 #endif
01708 #endif
01709
01710 /*
01711    we need, in exactly one source file, define the global struct that will hold
01712    the data we need to run utest. This macro allows the user to declare the
01713    data without having to use the UTEST_MAIN macro, thus allowing them to write
01714    their own main() function.
01715 */
01716 #define UTEST_STATE() struct utest_state_s utest_state = {0, 0, 0}
01717
01718 /*
01719    define a main() function to call into utest.h and start executing tests! A
01720    user can optionally not use this macro, and instead define their own main()
01721    function and manually call utest_main. The user must, in exactly one source
01722    file, use the UTEST_STATE macro to declare a global struct variable that
01723    utest requires.
01724 */
01725 #define UTEST_MAIN()                                                          \
01726   UTEST_STATE();                                                              \
01727   int main(int argc, const char *const argv[]) {                              \
01728     return utest_main(argc, argv);                                            \
01729   }
01730
01731 #endif /* SHEREDOM_UTEST_H_INCLUDED */
```

5.10 fixed_xor.c File Reference

Implementation of fixed-length buffer XOR helpers.

```
#include "fixed_xor.h"
#include <errno.h>
#include <limits.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for fixed_xor.c:



Macros

- #define FIXED_XOR_CHUNK 4096

Functions

- fixed_xor_status fixed_xor_buffers (const unsigned char *lhs, const unsigned char *rhs, unsigned char *out, size_t len)

  Implementation of fixed_xor_buffers().
- static fixed_xor_status fixed_xor_grow (unsigned char **buffer, size_t *capacity, size_t required)

  Ensure that `buffer` has capacity for `required` bytes.
- fixed_xor_status fixed_xor_stream (FILE *in, FILE *out)

  Implementation of fixed_xor_stream().
- const char * fixed_xor_status_string (fixed_xor_status status)

  Implementation of fixed_xor_status_string().

5.10.1 Detailed Description

Implementation of fixed-length buffer XOR helpers.

5.10.2 Macro Definition Documentation

5.10.2.1 FIXED_XOR_CHUNK

```
#define FIXED_XOR_CHUNK 4096
```

5.10.3 Function Documentation

5.10.3.1 fixed_xor_buffers()

```
fixed_xor_status fixed_xor_buffers (
            const unsigned char * lhs,
            const unsigned char * rhs,
            unsigned char * out,
            size_t len )
```

Implementation of fixed_xor_buffers().
XOR two buffers of equal length into an output buffer.

5.10.3.2 fixed_xor_grow()

```
static fixed_xor_status fixed_xor_grow (
            unsigned char ** buffer,
            size_t * capacity,
            size_t required )  [static]
```

Ensure that `buffer` has capacity for `required` bytes.

Parameters

buffer	Pointer to heap storage pointer to grow.
capacity	Current capacity in bytes (updated on success).
required	Target capacity required by the caller.

Returns

FIXED_XOR_OK on success or FIXED_XOR_ERR_OOM on allocation failure.

5.10.3.3 fixed_xor_status_string()

```
const char * fixed_xor_status_string (
            fixed_xor_status status )
```

Implementation of fixed_xor_status_string().
Convert a fixed_xor_status value into a human-readable string.

5.10.3.4 fixed_xor_stream()

```
fixed_xor_status fixed_xor_stream (
            FILE * in,
            FILE * out )
```

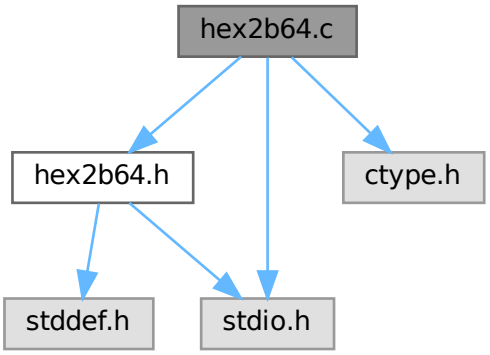
Implementation of fixed_xor_stream().
XOR two half-length buffers read sequentially from a stream.

5.11 hex2b64.c File Reference

Implementation of hex-to-Base64 conversion helpers.

```
#include "hex2b64.h"
#include <ctype.h>
#include <stdio.h>
```

Include dependency graph for hex2b64.c:



Functions

- static int hex_value (int c)

  Convert a single hexadecimal character into its integer value.
- static void encode_base64_chars (const unsigned char ∗in, size_t len, char encoded[4])

  Produce four Base64 characters from up to three bytes.
- static void encode_base64_block (const unsigned char ∗in, size_t len, FILE ∗out)

  Write a Base64 block derived from up to three input bytes.
- int hex2b64_stream (FILE ∗in, FILE ∗out)

  Implementation of hex2b64_stream().
- int hex2b64_buffer (const unsigned char ∗hex, size_t hex_len, unsigned char ∗out, size_t out_cap, size_t ∗out_len)

  Implementation of hex2b64_buffer().

Variables

- static const char b64_table [ ]

5.11.1 Detailed Description

Implementation of hex-to-Base64 conversion helpers.

5.11.2 Function Documentation

5.11.2.1 encode_base64_block()

```
static void encode_base64_block (
            const unsigned char * in,
            size_t len,
            FILE * out )  [static]
```

Write a Base64 block derived from up to three input bytes.

Parameters

in	Source bytes.
len	Number of source bytes.
out	Output stream that receives four Base64 characters.

5.11.2.2 encode_base64_chars()

```
static void encode_base64_chars (
            const unsigned char * in,
            size_t len,
            char encoded[4] )  [static]
```

Produce four Base64 characters from up to three bytes.

Parameters

in	Input bytes (padded implicitly with zeros).
len	Number of valid bytes (1-3).
encoded	Destination for the four encoded characters.

5.11.2.3 hex2b64_buffer()

```
int hex2b64_buffer (
```

```
            const unsigned char * hex,
            size_t hex_len,
            unsigned char * out,
            size_t out_cap,
            size_t * out_len )
```

Implementation of hex2b64_buffer().
Convert a memory buffer of hexadecimal characters into Base64.

5.11.2.4 hex2b64_stream()

```
int hex2b64_stream (
            FILE * in,
            FILE * out )
```

Implementation of hex2b64_stream().
Convert hexadecimal text read from a stream into Base64.

5.11.2.5 hex_value()

```
static int hex_value (
            int c )  [static]
```

Convert a single hexadecimal character into its integer value.

Parameters

c	Character to convert.

Returns

0-15 for valid hex digits, or -1 when `c` is not hexadecimal.

5.11.3 Variable Documentation

5.11.3.1 b64_table

```
const char b64_table[]  [static]
```

Initial value:

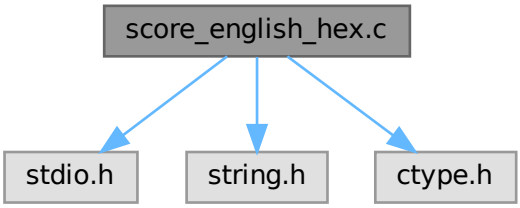
```
=
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    "abcdefghijklmnopqrstuvwxyz"
    "0123456789+/"
```

5.12 score_english_hex.c File Reference

Implementation of English scoring for hex-encoded strings.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

Include dependency graph for score_english_hex.c:



Functions

- static int hex_value (char c)

  Convert a single hex character to its integer value.
- double score_english_hex (const char *hex)

  Implementation of score_english_hex().

Variables

- static const double english_freq [27]

  English letter frequency proportions for a-z plus space.

5.12.1 Detailed Description

Implementation of English scoring for hex-encoded strings.

5.12.2 Function Documentation

5.12.2.1 hex_value()

```
static int hex_value (
            char c )  [static]
```

Convert a single hex character to its integer value.

Parameters

c	Hex digit to convert.

Returns

0-15 on success or -1 on invalid characters.

5.12.2.2 score_english_hex()

```
double score_english_hex (
            const char * hex )
```

Implementation of score_english_hex().
Score a hex string based on English letter frequency heuristics.

5.12.3 Variable Documentation

5.12.3.1 english_freq

```
const double english_freq[27]  [static]
```

Initial value:

```
= {
    0.0817,
    0.0150,
    0.0278,
    0.0425,
    0.1270,
    0.0223,
    0.0202,
    0.0609,
    0.0697,
    0.0015,
    0.0077,
    0.0403,
    0.0241,
    0.0675,
    0.0751,
    0.0193,
    0.0010,
    0.0599,
    0.0633,
    0.0906,
    0.0276,
    0.0098,
    0.0236,
    0.0015,
    0.0197,
    0.0007,
    0.1300
}
```

English letter frequency proportions for a-z plus space.
Index 0-25 correspond to 'a'-'z' and index 26 represents space.

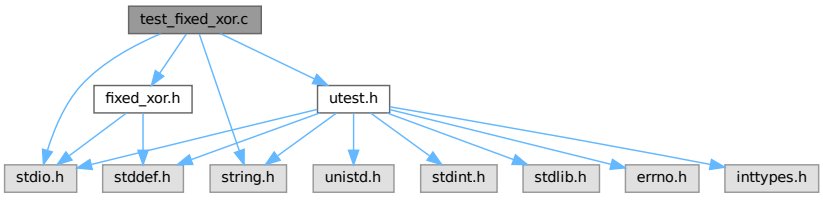
5.13 README.md File Reference

5.14 test_fixed_xor.c File Reference

Unit tests for fixed XOR helpers.

```
#include <stdio.h>
#include <string.h>
#include "fixed_xor.h"
#include "utest.h"
```

Include dependency graph for test_fixed_xor.c:



Functions

- UTEST (fixed_xor_buffers, simple_values)
- UTEST (fixed_xor_buffers, zero_length_is_ok)
- UTEST (fixed_xor_buffers, allows_output_alias)
- UTEST (fixed_xor_buffers, null_pointer_rejected)
- UTEST (fixed_xor_stream, processes_two_buffers)

- UTEST (fixed_xor_stream, empty_input_ok)
- UTEST (fixed_xor_stream, odd_length_input_fails)
- UTEST (fixed_xor_stream, large_input_triggers_growth)
- UTEST (fixed_xor_status_string, returns_text)
- UTEST_MAIN ()

5.14.1 Detailed Description

Unit tests for fixed XOR helpers.

5.14.2 Function Documentation

5.14.2.1 UTEST() [1/9]

```
UTEST (
            fixed_xor_buffers ,
            allows_output_alias  )
```

5.14.2.2 UTEST() [2/9]

```
UTEST (
            fixed_xor_buffers ,
            null_pointer_rejected  )
```

5.14.2.3 UTEST() [3/9]

```
UTEST (
            fixed_xor_buffers ,
            simple_values  )
```

5.14.2.4 UTEST() [4/9]

```
UTEST (
            fixed_xor_buffers ,
            zero_length_is_ok  )
```

5.14.2.5 UTEST() [5/9]

```
UTEST (
            fixed_xor_status_string ,
            returns_text  )
```

5.14.2.6 UTEST() [6/9]

```
UTEST (
            fixed_xor_stream ,
            empty_input_ok  )
```

5.14.2.7 UTEST() [7/9]

```
UTEST (
            fixed_xor_stream ,
            large_input_triggers_growth  )
```

5.14.2.8 UTEST() [8/9]

```
UTEST (
            fixed_xor_stream ,
            odd_length_input_fails  )
```

5.14.2.9 UTEST() [9/9]

```
UTEST (
            fixed_xor_stream ,
            processes_two_buffers  )
```

5.14.2.10 UTEST_MAIN()

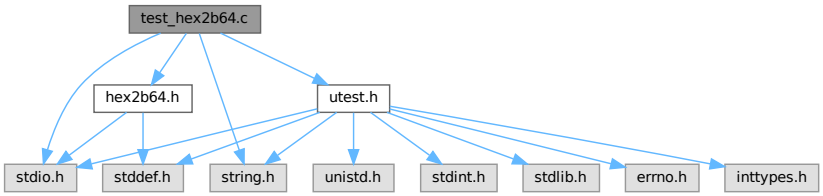
```
UTEST_MAIN ( )
```

5.15 test_hex2b64.c File Reference

Unit tests for the hex2b64 conversion helpers.

```
#include <stdio.h>
#include <string.h>
#include "hex2b64.h"
#include "utest.h"
```

Include dependency graph for test_hex2b64.c:



Functions

- static int convert_hex_string (const char *hex, char *out, size_t out_cap)

  Run hex2b64_stream() over an in-memory string.
- static int convert_hex_buffer (const char *hex, unsigned char *out, size_t out_cap, size_t *out_len)

  Convenience wrapper around hex2b64_buffer() for literals.
- UTEST (stream, hello_world_plain)
- UTEST (stream, ignores_whitespace)
- UTEST (stream, mixed_case_hex)
- UTEST (stream_edge, empty_input)
- UTEST (stream_edge, whitespace_only)
- UTEST (stream_edge, single_byte)
- UTEST (stream_edge, two_bytes)
- UTEST (stream_edge, three_bytes)
- UTEST (stream_edge, odd_number_of_digits_is_error)
- UTEST (stream_edge, invalid_char_in_middle)
- UTEST (stream_edge, invalid_first_char)
- UTEST (stream_edge, many_bytes_sanity)
- UTEST (buffer, hello_world_plain)
- UTEST (buffer, ignores_whitespace)
- UTEST (buffer, empty_input)
- UTEST (buffer_edge, odd_digits_error)
- UTEST (buffer_edge, invalid_character)
- UTEST (buffer_edge, insufficient_output_capacity)
- UTEST_MAIN ()

5.15.1 Detailed Description

Unit tests for the hex2b64 conversion helpers.

5.15.2 Function Documentation

5.15.2.1 convert_hex_buffer()

```
static int convert_hex_buffer (
            const char * hex,
            unsigned char * out,
            size_t out_cap,
            size_t * out_len )  [static]
```

Convenience wrapper around hex2b64_buffer() for literals.

Parameters

hex	Null-terminated hex literal.
out	Destination buffer.
out_cap	Capacity of `out`.
out_len	Optional pointer receiving bytes written.

Returns

0 on success, non-zero on failure.

5.15.2.2 convert_hex_string()

```
static int convert_hex_string (
            const char * hex,
            char * out,
            size_t out_cap )  [static]
```

Run hex2b64_stream() over an in-memory string.

Parameters

hex	Null-terminated hex input.
out	Destination buffer for Base64 text plus newline.
out_cap	Capacity of `out`.

Returns

0 on success or negative on setup error.

5.15.2.3 UTEST() [1/18]

```
UTEST (
            buffer ,
            empty_input  )
```

5.15.2.4 UTEST() [2/18]

```
UTEST (
            buffer ,
            hello_world_plain  )
```

5.15.2.5 UTEST() [3/18]

```
UTEST (
            buffer ,
            ignores_whitespace  )
```

5.15.2.6 UTEST() [4/18]

```
UTEST (
            buffer_edge ,
            insufficient_output_capacity  )
```

5.15.2.7 UTEST() [5/18]

```
UTEST (
            buffer_edge ,
            invalid_character  )
```

5.15.2.8 UTEST() [6/18]

```
UTEST (
            buffer_edge ,
            odd_digits_error  )
```

5.15.2.9 UTEST() [7/18]

```
UTEST (
            stream ,
            hello_world_plain  )
```

5.15.2.10 UTEST() [8/18]

```
UTEST (
            stream ,
            ignores_whitespace  )
```

5.15.2.11 UTEST() [9/18]

```
UTEST (
            stream ,
            mixed_case_hex  )
```

5.15.2.12 UTEST() [10/18]

```
UTEST (
            stream_edge ,
            empty_input  )
```

5.15.2.13 UTEST() [11/18]

```
UTEST (
            stream_edge ,
            invalid_char_in_middle  )
```

5.15.2.14 UTEST() [12/18]

```
UTEST (
            stream_edge ,
            invalid_first_char  )
```

5.15.2.15 UTEST() [13/18]

```
UTEST (
            stream_edge ,
            many_bytes_sanity  )
```

5.15.2.16 UTEST() [14/18]

```
UTEST (
            stream_edge ,
            odd_number_of_digits_is_error  )
```

5.15.2.17 UTEST() [15/18]

```
UTEST (
            stream_edge ,
            single_byte  )
```

5.15.2.18 UTEST() [16/18]

```
UTEST (
            stream_edge ,
            three_bytes  )
```

5.15.2.19 UTEST() [17/18]

```
UTEST (
            stream_edge ,
            two_bytes  )
```

5.15.2.20 UTEST() [18/18]

```
UTEST (
            stream_edge ,
            whitespace_only  )
```

5.15.2.21 UTEST_MAIN()

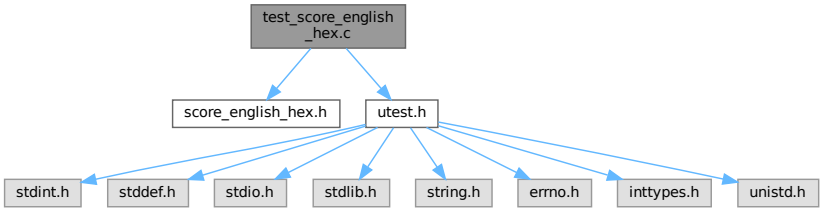
```
UTEST_MAIN ( )
```

5.16 test_score_english_hex.c File Reference

Unit tests for the English scoring heuristic.

```
#include "score_english_hex.h"
#include "utest.h"
```

Include dependency graph for test_score_english_hex.c:



Functions

- UTEST (score_english_hex, english_phrase_beats_random)
- UTEST (score_english_hex, rejects_invalid_hex_character)
- UTEST (score_english_hex, rejects_odd_length)
- UTEST (score_english_hex, rejects_non_printable_bytes)
- UTEST_MAIN ()

5.16.1 Detailed Description

Unit tests for the English scoring heuristic.

5.16.2 Function Documentation

5.16.2.1 UTEST() [1/4]

```
UTEST (
            score_english_hex ,
            english_phrase_beats_random  )
```

5.16.2.2 UTEST() [2/4]

```
UTEST (
            score_english_hex ,
            rejects_invalid_hex_character  )
```

5.16.2.3 UTEST() [3/4]

```
UTEST (
            score_english_hex ,
            rejects_non_printable_bytes  )
```

5.16.2.4 UTEST() [4/4]

```
UTEST (
            score_english_hex ,
            rejects_odd_length  )
```

5.16.2.5 UTEST_MAIN()

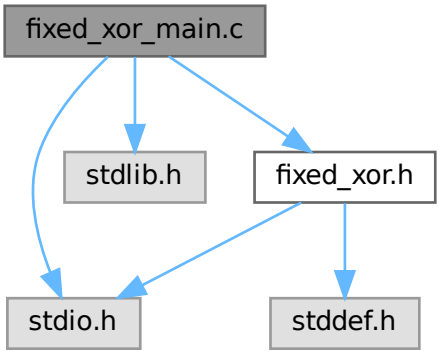
```
UTEST_MAIN ( )
```

5.17 fixed_xor_main.c File Reference

Command-line wrapper for XORing two equal-length buffers.

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

Include dependency graph for fixed_xor_main.c:



Functions

- int main (void)

  Entry point that streams stdin through fixed_xor_stream().

5.17.1 Detailed Description

Command-line wrapper for XORing two equal-length buffers.

5.17.2 Function Documentation

5.17.2.1 main()

```
int main (
            void  )
```

Entry point that streams stdin through fixed_xor_stream().

Returns

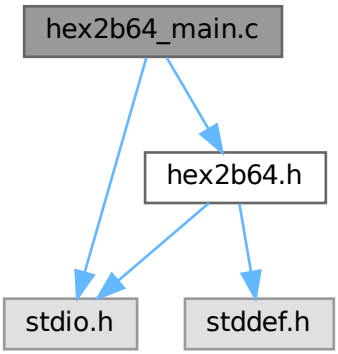
EXIT_SUCCESS on success, EXIT_FAILURE otherwise.

5.18 hex2b64_main.c File Reference

Command-line tool that converts hex input to Base64.

```
#include <stdio.h>
#include "hex2b64.h"
```

Include dependency graph for hex2b64_main.c:



Functions

- int main (void)

  Entry point that wires stdin/stdout through hex2b64_stream().

5.18.1 Detailed Description

Command-line tool that converts hex input to Base64.

5.18.2 Function Documentation

5.18.2.1 main()

```
int main (
            void  )
```

Entry point that wires stdin/stdout through hex2b64_stream().

Returns

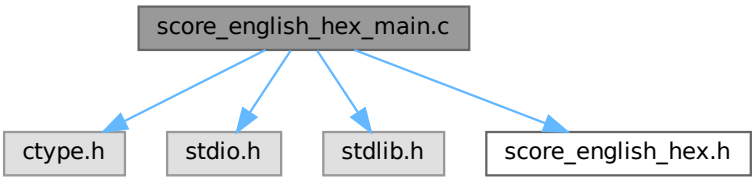
0 on success, non-zero if conversion fails.

5.19 score_english_hex_main.c File Reference

CLI utility that scores hex data for English-likeness.

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

```
#include "score_english_hex.h"
```
Include dependency graph for score_english_hex_main.c:



Functions

- static double score_to_percentage (double raw)

  Map a raw English score to a 0-100 percentage.
- int main (void)

  Read hex from stdin, score it, and print a percentage.

5.19.1 Detailed Description

CLI utility that scores hex data for English-likeness.

5.19.2 Function Documentation

5.19.2.1 main()

```
int main (
            void  )
```
Read hex from stdin, score it, and print a percentage.

Returns

EXIT_SUCCESS on valid input, EXIT_FAILURE otherwise.

5.19.2.2 score_to_percentage()

```
static double score_to_percentage (
            double raw )  [static]
```
Map a raw English score to a 0-100 percentage.

Parameters

raw	Score returned by score_english_hex().

Returns

Normalized percentage.

Index