API for Block-Compression Test

Interface

```
#pragma once

#include <stdint.h>

#ifdef _WIN32
#define CDECL __cdecl
#else
#define CDECL
#endif

#ifdef __cplusplus
extern "C" {
#endif

typedef unsigned char BYTE;

/**
*****************************************************************************
* \brief Initialize encoding
*
* \param cmprContext          [OUT] - pointer to initialized compressor
context pointer
*
* \return 0 if OK
*/
int32_t CDECL encodeInit( void **cmprContext );

/**
*****************************************************************************
* \brief Run encoding
*
* \param inSize               [IN] - input data block size (assume 2^16)
* \param inPtr                [IN] - pointer to input data block
* \param outSize              [OUT] - pointer to output block size
* \param outPtr               [OUT] - pointer to output block
* \param cmprContext          [IN] - pointer to previously initialized
compressor context
*
* \return 0 if OK
*/
int32_t CDECL encodeRun( int32_t inSize, const BYTE *inPtr, int32_t *outSize,
BYTE *outPtr, void *cmprContext );

/**
*****************************************************************************
* \brief Initialize decoding
*
```

```
* \param cmprContext          [OUT] - pointer to initialized compressor
context pointer
*
* \return 0 if OK
*/
int32_t CDECL decodeInit( void **cmprContext );

/**
*****************************************************************************
* \brief Run decoding
*
* \param inSize               [IN] - input (compressed) data block size
* \param inPtr                [IN] - pointer to input data block
* \param subblkIdx            [IN] - index of 8 KiB subblock, 0..7
* \param outSize              [OUT] - pointer to output subblock size
(normally 2^13 bytes)
* \param outPtr               [OUT] - pointer to output block
* \param cmprContext          [IN] - pointer to previously initialized
compressor context
*
* \return 0 if OK
*/
int32_t CDECL decodeRun( int32_t inSize, const BYTE *inPtr, int32_t subblkIdx,
int32_t *outSize, BYTE *outPtr, void *cmprContext );

#ifdef __cplusplus
}
#endif
```

Examples

We have published 2 examples of application of this interface to existing compressors (files needed for the final linking of the binary files are not included).
Zstd: https://gdcc.tech/wp-content/uploads/Zstd_gdcc_T4.zip
Brotli: https://gdcc.tech/wp-content/uploads/brotli_gdcc_T4.zip

Testbench

We have published the source code of a program, that can help you with the testing of your libraries. The actual testing for competition uses similar, but somewhat different program.

Testbench can be seen here: https://gdcc.tech/wp-content/uploads/testbench_gdcc_T4.zip

Description

The compression library takes as input an arbitrary block of input data. The size of each block is 64 KiB (65536 bytes) for this year's competition, and every block contains eight consecutive subblocks of 8 KiB (8,192 bytes) each. The library compresses a given block to allow fast decompression of the constituent subblocks. The decompression library (or the same

compression library operating in decoding mode) takes as input an arbitrary compressed block, whose size before compression was 64 KiB, and the index (number) of the 8 KiB subblock that requires decoding. The first subblock in each block has index `subblkIdx` = 0; the last one has index `subblkIdx` = 7. To avoid simple caching, the decoding process runs eight times such that for each run, one and only one subblock undergoes decoding in each block. The order of subblock selection in each block is random, but all subblocks are ultimately decoded exactly once.

The test bench implements the following behavior:

Compression:

- Create a separate process for encoding
- Initialize encoding by calling `encodeInit()`
- For each randomly chosen input data block (every block is processed once and only once):
- Call `encodeRun()` and remember the output compressed data
- Save all compressed data
- Stop the process

Decompression:

- Run eight times so that for each run, only one subblock from each original block is decoded:
  - Create a separate process for decoding
  - Initialize decoding by calling `decodeInit()`
  - For each randomly chosen compressed data block and its lone, randomly chosen but yet-to-be decompressed subblock (every subblock is decompressed exactly once during the decompression process):
    - Call `decodeRun()` for the subblock and remember the output uncompressed data
  - Stop the process
- Check whether the decompression is lossless

In the case of separate compression and decompression libraries, each should implement the respective functions—that is, `encodeInit()` and `encodeRun()` for compression, and `decodeInit()` and `decodeRun()` for decompression.

If any library function returns a nonzero value, we treat it as a critical error and terminate all processes.

The test bench allocates and deallocates the input buffer `inPtr` and output buffer `outPtr` independently. The size of these buffers for encoding is the test's block size (`inSize` = 65,536 bytes). The size of the output buffer `outPtr` for decoding is 8,192 bytes.

If the library is unable to compress a given 64 KiB block—that is, the block's compressed size exceeds 65,536 bytes or is equal to 0 bytes—the test bench assumes the size is 65,537 bytes (including 1 byte for the data-copy flag). It will process such blocks on its own during decompression by copying the original block data. The total decompression time will include the time required for this copy step.

Note that the test bench passes the size of a given compressed 64 KiB block to the decompressor; storing this information in the compressed block is unnecessary. Also, this information is disregarded when computing the overall compressed-data size for the block-compression test.

To optimize the time expense for frequent calls to memory-allocation functions, we recommend using static memory allocation inside the library.