Global Data Compression Competition

https://globalcompetition.compression.ru

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
* \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 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 decodeRun( int32_t inSize, const BYTE *inPtr, int32_t *outSize,
BYTE *outPtr, void *cmprContext );

#ifdef __cplusplus
}
#endif
```

Description

The test bench implements the following behavior:
Compression:

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

Decompression:

- Create a separate process for decoding
- Initialize decoding, `decodeInit()`
- For each randomly chosen compressed data block (every block is processed once and only once):
  - Call `decodeRun()`, remember the output uncompressed data
- Check that the decompression is lossless
- Stop the process

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

If any library function returns a nonzero value, we treat it as a critical error and stop any further processing.

The test bench allocates and deallocates the input `inPtr` and output buffer `outPtr` on its own. The size of the input buffer is the block size to be used in the test (`inSize =` 32,768 bytes). The output buffer size is the block size + 2000 bytes, giving 34,768 bytes. Thus the compressor should control that the size of the compressed (encoded) data does not exceed this threshold `inSize + 2000`.

If the library is unable to compress a given block—that is, the block's compressed size exceeds 32,768 bytes or is equal to 0 bytes—the test bench assumes the size is 32,769 bytes (including 1 byte for the data-copy flag). The test bench 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 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.