Contents

1 include/binary_io/binary_io.hpp 2

2 include/integer_codes/elias_delta.hpp 5

3 include/integer_codes/elias_gamma.hpp 6

4 include/integer_codes/golomb_rice.hpp 7

5 include/integer_codes/integer_codes.hpp 8

6 include/integer_codes/truncated_binary.hpp 9

7 include/integer_codes/unary.hpp 11

8 include/integer_codes/varint.hpp 12

9 include/integer_codes/zigzag.hpp 13

10 include/toy_compression.hpp 15

11 test/binary_io_test.hpp 16

12 test/elias_delta_test.hpp 19

13 test/elias_gamma_test.hpp 20

14 test/golomb_rice_test.hpp 21

15 test/test.cpp 23

16 test/truncated_binary_test.hpp 24

17 test/unary_test.hpp 28

18 test/varint_test.hpp 29

19 test/zigzag_test.hpp 30

20 toy_test/toy_test.hpp 31

1 include/binary_io/binary_io.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

namespace binary_io {
   template <typename Iterator, typename Iterator2>
   struct bit_reader {
      Iterator      pos;
      Iterator2     end;
      std::uint8_t  buf;
      std::uint64_t total_count;
      unsigned int  bits_left;

      bit_reader( Iterator&& begin_, Iterator2&& end_ ) :
         pos{begin_},
         end{end_},
         buf{0},
         total_count{0},
         bits_left{0} {}
      bit_reader( bit_reader const& )  = delete;
      bit_reader( bit_reader const&& ) = delete;
      bit_reader& operator=( bit_reader const& ) = delete;
      bit_reader& operator=( bit_reader const&& ) = delete;
      ~bit_reader()                               = default;

      void input_byte() {
         if ( pos == end ) {
            throw std::out_of_range(
               "Attempt to read bits beyond end of range" );
         }
         bits_left = std::numeric_limits<std::uint8_t>::digits;
         buf       = *pos;
         ++pos;
      }

      bool read_bit() {
         if ( bits_left == 0 ) {
            input_byte();
         }
         ++total_count;
         --bits_left;
         return ( buf >> bits_left ) & 1;
      }

      template <typename T>
```

```
   T read_bits( unsigned int bits ) {
      using UT = std::make_unsigned_t<T>;
      UT temp{0};
      for ( unsigned int i = 0; i < bits; i++ ) {
         temp = ( temp << 1 ) | read_bit();
      }
      return temp;
   }
};

template <typename Iterator, typename Iterator2,
          typename
          = typename std::iterator_traits<Iterator>::iterator_category>
auto make_bit_reader( Iterator&& begin, Iterator2&& end )
   -> bit_reader<Iterator, Iterator2> {
   return bit_reader<Iterator, Iterator2>{std::forward<Iterator>( begin ),
                                          std::forward<Iterator2>( end )};
}

template <typename Container,
          typename = decltype( std::declval<Container>().cbegin() )>
auto make_bit_reader( Container const& c ) {
   return make_bit_reader( c.cbegin(), c.cend() );
}

template <typename stream_t>
auto make_bit_reader( stream_t& stream ) -> decltype(
   make_bit_reader( std::istream_iterator<unsigned char>( stream ),
                    std::istream_iterator<unsigned char>() ) ) {
   return make_bit_reader( std::istream_iterator<unsigned char>( stream ),
                           std::istream_iterator<unsigned char>() );
}

template <typename Iterator>
struct bit_writer {
   Iterator      pos;
   std::uint8_t  buf;
   std::uint64_t total_count;
   unsigned int  bits_left;
   bit_writer( Iterator&& begin_ ) :
      pos{std::forward<Iterator>( begin_ )},
      buf{0},
      total_count{0},
      bits_left{std::numeric_limits<std::uint8_t>::digits} {}
   ~bit_writer() {
      if ( bits_left != std::numeric_limits<std::uint8_t>::digits ) {
         flush();
      }
   }
   bit_writer( bit_writer const& )  = delete;
   bit_writer( bit_writer const&& ) = delete;
   bit_writer& operator=( bit_writer const& ) = delete;
   bit_writer& operator=( bit_writer const&& ) = delete;

   void output_byte() {
      *pos = buf;
      buf  = 0;
```

```
            ++pos;
            bits_left = std::numeric_limits<std::uint8_t>::digits;
        }

        void write_bit( bool bit ) {
            --bits_left;
            ++total_count;
            buf |= ( static_cast<std::uint8_t>( bit ) << bits_left );
            if ( bits_left == 0 ) {
                output_byte();
            }
        }

        template <typename T, typename UT = std::make_unsigned_t<T>>
        void write_bits( T value, unsigned int num_bits ) {
            for ( int i = num_bits; i > 0; --i ) {
                UT mask = static_cast<UT>( 1 ) << ( i - 1 );
                write_bit( ( static_cast<UT>( value ) & mask ) != 0 );
            }
        }

        void flush() { output_byte(); }
    };

    template <typename Iterator,
              typename
              = typename std::iterator_traits<Iterator>::iterator_category>
    auto make_bit_writer( Iterator&& begin ) -> bit_writer<Iterator> {
        return bit_writer<Iterator>{std::forward<Iterator>( begin )};
    }

    template <typename Container,
              typename = decltype( std::declval<Container>().begin() )>
    auto make_bit_writer( Container&& c ) {
        return make_bit_writer( std::back_inserter( c ) );
    }

    template <typename stream_t>
    auto make_bit_writer( stream_t& stream ) -> decltype(
        make_bit_writer( std::ostream_iterator<unsigned char>( stream ) ) ) {
        return make_bit_writer( std::ostream_iterator<unsigned char>( stream ) );
    }
} // namespace binary_io
```

2 include/integer_codes/elias_delta.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

struct elias_delta {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( elias_delta::encode, x >= 1 );
      auto b = 1 + static_cast<T>( std::floor( std::log2( x ) ) );
      elias_gamma::template encode<T>( b, storage );
      storage.template write_bits<T>( ( x - ( 1 << ( b - 1 ) ) ), b - 1 );
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      auto b = elias_gamma::template decode<T>( storage );
      auto x = storage.template read_bits<T>( b - 1 );
      return ( 1 << ( b - 1 ) ) + x;
   }
};
```

3 include/integer_codes/elias_gamma.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

struct elias_gamma {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( elias_gamma::encode, x >= 1 );
      auto b = 1 + static_cast<T>( std::floor( std::log2( x ) ) );
      unary::template encode<T>( b, storage );
      storage.write_bits( x - ( 1 << ( b - 1 ) ), b - 1 );
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      auto b = unary::template decode<T>( storage );
      auto x = storage.template read_bits<T>( b - 1 );
      return ( 1 << ( b - 1 ) ) + x;
   }
};
```

4 include/integer_codes/golomb_rice.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

struct golomb {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, T b, binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( golomb::encode, x >= 1 && b >= 1 );
      auto q = ( x - 1 ) / b;
      auto r = ( x - 1 ) % b;
      unary::template encode<T>( q + 1, storage );
      truncated_binary::template encode<T>( r, b, storage );
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( T b, binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      TOY_COMPRESSION_ASSERT( golomb::decode, b >= 1 );
      auto q = unary::template decode<T>( storage ) - 1;
      auto r = truncated_binary::template decode<T>( b, storage );
      return r + q * b + 1;
   }
};

struct rice {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, T k, binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( rice::encode, x >= 1 );
      auto b = static_cast<T>( 1 ) << k;
      golomb::encode( x, b, storage );
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( T k, binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      auto b = static_cast<T>( 1 ) << k;
      return golomb::template decode<T>( b, storage );
   }
};
```

5 include/integer_codes/integer_codes.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

namespace integer_codes {
   template <typename T>
   using unsigned_of = typename std::make_unsigned_t<T>;

   template <typename T>
   using signed_of = typename std::make_signed_t<T>;

#include "unary.hpp"
#include "truncated_binary.hpp"
#include "elias_gamma.hpp"
#include "elias_delta.hpp"
#include "golomb_rice.hpp"
#include "varint.hpp"
#include "zigzag.hpp"

} // namespace integer_codes
```

6 include/integer_codes/truncated_binary.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

struct truncated_binary {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, T n, binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( truncated_binary::encode, n >= 1 && x <= n );
      auto k      = static_cast<T>( std::floor( std::log2( n ) ) );
      auto u      = ( static_cast<T>( 1 ) << ( k + 1 ) ) - n;
      bool lesser = x < u;
      x           = lesser ? x : x + u;
      k           = lesser ? k : k + 1;
      storage.write_bits( x, k );
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( T n, binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      TOY_COMPRESSION_ASSERT( truncated_binary::decode, n >= 1 );
      auto k          = static_cast<T>( std::floor( std::log2( n ) ) );
      auto u          = ( static_cast<T>( 1 ) << ( k + 1 ) ) - n;
      auto x          = storage.template read_bits<T>( k );
      bool greater_eq = x >= u;
      x = greater_eq ? ( ( x << 1 ) | storage.read_bit() ) - u : x;
      return x;
   }
};

struct centered_truncated_binary {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, T n, binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( centered_truncated_binary, n >= 1 );
      TOY_COMPRESSION_ASSERT( centered_truncated_binary,
                              ( 1 <= x ) && ( x <= n ) );

      auto const top = static_cast<T>( 1 )
                       << static_cast<T>( std::ceil( std::log2( n ) ) );
      auto const offset   = n - ( top >> 1 );
      auto const centered = ( n + x - offset ) % n;
      truncated_binary::encode( centered, n, storage );
   }
```

```
   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( T n, binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      TOY_COMPRESSION_ASSERT( centered_truncated_binary, n >= 1 );
      auto const top = static_cast<T>( 1 )
                       << static_cast<T>( std::ceil( std::log2( n ) ) );
      auto const offset            = n - ( top >> 1 );
      auto const partially_decoded = truncated_binary::decode( n, storage );
      auto       decoded           = ( partially_decoded + offset ) % n;
      return decoded;
   }
};

struct binary_in_range {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, T low, T high,
                       binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( binary_in_range, low >= 1 && low < high );
      TOY_COMPRESSION_ASSERT( binary_in_range, x >= low && x <= high );
      centered_truncated_binary::encode( x - low + 1, high - low + 1, storage );
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( T low, T high,
                    binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      TOY_COMPRESSION_ASSERT( binary_in_range, low >= 1 );
      TOY_COMPRESSION_ASSERT( binary_in_range, low < high );
      return centered_truncated_binary::decode( high - low + 1, storage ) + low
             - 1;
   }
};
```

7 include/integer_codes/unary.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

struct unary {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, binary_io::bit_writer<Iterator>& storage ) {
      TOY_COMPRESSION_ASSERT( unary::encode, x > 0 );
      T temp{x};
      while ( temp > 1 ) {
         --temp;
         storage.write_bit( 0 );
      }
      storage.write_bit( 1 );
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      T temp{1};
      while ( !storage.read_bit() ) {
         ++temp;
      }
      return temp;
   }
};
```

8 include/integer_codes/varint.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

struct varint {
   template <typename T, typename Iterator,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static void encode( T x, binary_io::bit_writer<Iterator>& storage ) {
      constexpr auto digits = std::numeric_limits<T>::digits;
      // a varint can't get bigger than this, really
      constexpr auto bytes_to_reserve
         = static_cast<std::size_t>( digits + 6 ) / 7;
      std::array<std::uint8_t, bytes_to_reserve> temp_buffer{};
      auto pos{bytes_to_reserve - 1}; // last byte
      gsl::at( temp_buffer, pos-- ) = static_cast<std::uint8_t>( x & 127 );

      while ( x >>= 7 ) {
         --x;
         gsl::at( temp_buffer, pos-- )
            = static_cast<std::uint8_t>( 128 | ( x & 127 ) );
      }
      for ( pos++; pos < bytes_to_reserve; pos++ ) {
         storage.template write_bits<std::uint8_t>( gsl::at( temp_buffer, pos ),
                                                    8 );
      }
   }

   template <typename T, typename Iterator, typename Iterator2,
             typename = std::enable_if_t<std::is_unsigned_v<T>>>
   static T decode( binary_io::bit_reader<Iterator, Iterator2>& storage ) {
      T            output_val       = 0;
      std::uint8_t continuation_val = 0;
      do {
         auto val          = storage.template read_bits<std::uint8_t>( 8 );
         continuation_val = val & 0x80;
         val &= 0x7f;
         if ( continuation_val ) {
            ++val;
         }
         output_val = ( output_val << 7 ) + val;
      } while ( continuation_val );
      return output_val;
   }
};
```

9 include/integer_codes/zigzag.hpp

```cpp
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once

namespace {
   template <typename T>
   using enable_enc
      = std::enable_if_t<std::is_integral_v<T> && std::is_signed_v<T>>;

   template <typename T>
   using enable_dec
      = std::enable_if_t<std::is_integral_v<T> && std::is_unsigned_v<T>>;
} // namespace

struct zigzag {
   template <typename T, typename = enable_enc<T>>
   constexpr static auto encode( T x ) -> unsigned_of<T> {
      return ( static_cast<unsigned_of<T>>( std::abs( x ) ) << 1 )
             | static_cast<unsigned_of<T>>( x <= 0 );
   }

   template <typename T, typename = enable_dec<T>>
   constexpr static auto decode( const T x ) -> signed_of<T> {
#define BIT_HACK
#ifdef BIT_HACK
      const signed_of<T> sign
         = -static_cast<signed_of<T>>( ( x & 1 ) && ( x != 1 ) );
      const signed_of<T> magnitude = x >> 1;
      return ( magnitude + sign ) ^ sign;
#else
      const T            sign      = ( x & 1 ) && ( x != 1 );
      const signed_of<T> magnitude = x >> 1;
      return sign ? -magnitude : magnitude;
#endif // BIT_HACK
#undef BIT_HACK
   }
};

struct offset_zigzag {
   template <typename T, typename = enable_enc<T>>
   constexpr static auto encode( T x, T offset ) -> unsigned_of<T> {
      return zigzag::encode( x - offset );
   }

   template <typename T, typename = enable_dec<T>>
```

```
    constexpr static auto decode( const T x, const signed_of<T> offset )
        -> signed_of<T> {
        return zigzag::decode( x ) + offset;
    }
};
```

10 include/toy_compression.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once
#ifndef TOY_COMPRESSION_HPP_INCLUDED
#define TOY_COMPRESSION_HPP_INCLUDED

#define stringify2( x ) #x
#define stringify( x ) stringify2( x )
#define TOY_COMPRESSION_ASSERT( coder, condition )                \
   if ( !( condition ) ) {                                        \
      throw std::invalid_argument( #coder __FILE__ ":" stringify( \
         __LINE__ ) " => exception(" #condition ") failed." );    \
   }

#include "gsl/gsl"
#include <array>
#include <cmath>
#include <cstdint>
#include <exception>
#include <iostream>
#include <iterator>
#include <limits>
#include <type_traits>
#include <utility>

namespace toy_compression {

#include "binary_io/binary_io.hpp"
#include "integer_codes/integer_codes.hpp"

} // namespace toy_compression
#endif // TOY_COMPRESSION_HPP_INCLUDED
```

11 test/binary_io_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

toy_test::suite const reader_suite{
   "Test for bit_reader",
   {
      {"bit_reader can be created from a pair of iterators",
       [] {
          container c;
          (void)make_bit_reader( c.begin(), c.end() );
          ASSERT( true );
       }},

      {"bit_reader can be created from a container",
       [] {
          container c;
          (void)make_bit_reader( c );
          ASSERT( true );
       }},

#ifdef DISABLE_TO_AVOID_WARNING
      {"bit_reader can be created from an istream",
       [] {
          std::istringstream ss;
          (void)make_bit_reader( ss );
          ASSERT( true );
       }},
#endif // DISABLE_TO_AVOID_WARNING

      {"bit_reader throws exception when read from empty",
       [] {
          container c{};
          auto      reader = make_bit_reader( c );
          THROWS( reader.read_bit(), std::out_of_range );
       }},

      {"bit_reader.read_bit() returns the first available bit",
       [] {
          container c{0x70};
          auto      reader = make_bit_reader( c );
          ASSERT( reader.read_bit() == 0 );
       }},

      {"bit_reader.read_bits() returns multiple bits",
       [] {
          container c{0x70, 0x0f, 0x0};
```

```
            auto      reader = make_bit_reader( c );
            (void)reader.read_bit();
            auto temp = reader.read_bits<test_type>( 16 );
            ASSERT( temp == 0xe01e );
         }},

   }};

toy_test::suite const writer_suite{
   "Test for bit_writer",
   {
      {"bit_writer can be created from an iterator",
       [] {
          container c;
          (void)make_bit_writer( std::back_inserter( c ) );
          ASSERT( true );
       }},

      {"bit_writer can be created from a container",
       [] {
          container c;
          (void)make_bit_writer( c );
          ASSERT( true );
       }},

#ifdef DISABLE_TO_AVOID_WARNING
      {"bit_writer can be created from an ostream",
       [] {
          std::ostringstream ss;
          {
             auto writer = make_bit_writer( ss );
             for ( int i = 0; i < 10; i++ ) {
                writer.write_bit( true );
             }
          }
          { (void)make_bit_writer( ss ); }

          ASSERT( true );
       }},
#endif // DISABLE_TO_AVOID_WARNING

      {"bit_writer.write_bit() writes a bit to the buffer",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          writer.write_bit( true );
          writer.write_bit( true );
          writer.write_bit( true );
          writer.write_bit( false );
          writer.flush();
          ASSERT( c[0] == 0xe0 );
       }},

      {"bit_writer.write_bits() writes a group of bits to the buffer",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
```

```
        writer.write_bits<test_type>( 0xfaf, 12 );
        writer.flush();
        ASSERT( ( c[0] == 0xfa ) && ( c[1] == 0xf0 ) );
     }},

    {"bit_writer␣flushes␣partial␣bytes␣to␣the␣buffer",
     [] {
        container c;
        {
           auto writer = make_bit_writer( c );
           writer.write_bit( true );
           writer.write_bit( true );
        }
        ASSERT( c[0] == 0xc0 );
     }},
};
```

12 test/elias_delta_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

void test_elias_delta_coder( test_type value ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::elias_delta::encode( value, writer );
   }
   auto reader  = make_bit_reader( c );
   auto decoded = integer_codes::elias_delta::decode<test_type>( reader );
   ASSERT( decoded == value );
}

toy_test::suite elias_delta_suite{
   "Test for elias delta coder",
   {
      {"throws an exception for x=0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::elias_delta::encode<test_type>( 0, writer ),
                  std::invalid_argument );
       }},

      {"encodes a value of 1", [] { test_elias_delta_coder( 1 ); }},
      {"encodes a value of 3", [] { test_elias_delta_coder( 3 ); }},
   }};
```

13 test/elias_gamma_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

void test_elias_gamma_coder( test_type value ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::elias_gamma::encode( value, writer );
   }
   auto reader  = make_bit_reader( c );
   auto decoded = integer_codes::elias_gamma::decode<test_type>( reader );
   ASSERT( decoded == value );
}

toy_test::suite elias_gamma_suite{
   "Test for elias gamma coder",
   {
      {"throws an exception for x=0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::elias_gamma::encode<test_type>( 0, writer ),
                  std::invalid_argument );
       }},
      {"encodes a value of 2", [] { test_elias_gamma_coder( 2 ); }},
      {"encodes a value of 3", [] { test_elias_gamma_coder( 3 ); }},
   }};
```

14 test/golomb_rice_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

void test_golomb( test_type x, test_type b ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::golomb::encode( x, b, writer );
   }
   auto reader = make_bit_reader( c );
   auto result = integer_codes::golomb::template decode<test_type>( b, reader );
   ASSERT( result == x );
}

void test_rice( test_type x, test_type b ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::rice::encode( x, b, writer );
   }
   auto reader = make_bit_reader( c );
   auto result = integer_codes::rice::template decode<test_type>( b, reader );
   ASSERT( result == x );
}

toy_test::suite golomb_suite{
   "Test␣for␣golomb␣coder",
   {
      {"encode␣throws␣an␣exception␣for␣x=0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::golomb::encode<test_type>( 0, 5, writer ),
                  std::invalid_argument );
       }},

      {"encode␣throws␣an␣exception␣for␣b=0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::golomb::encode<test_type>( 5, 0, writer ),
                  std::invalid_argument );
       }},

      {"decode␣throws␣an␣exception␣for␣b=0",
       [] {
```

```
            container c( 100 );
            auto      reader = make_bit_reader( c );
            THROWS( integer_codes::golomb::decode<test_type>( 0, reader ),
                    std::invalid_argument );
         }},

         {"encodes␣and␣decodes␣small␣integers",
          [] {
            for ( test_type i = 1; i < 256; i++ ) {
               for ( test_type b = 1; b < 256; b++ ) {
                  test_golomb( i, b );
               }
            }
         }},
   }};

toy_test::suite rice_suite{
   "Test␣for␣rice␣coder",
   {
      {"throws␣an␣exception␣for␣x=0",
       [] {
          container c( 100 );
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::rice::encode<test_type>( 0, 4, writer ),
                  std::invalid_argument );
       }},

      {"encodes␣and␣decodes␣small␣integers",
       [] {
          for ( test_type i = 1; i < 256; i++ ) {
             for ( test_type k = 0; k < 16; k++ ) {
                test_rice( i, k );
             }
          }
       }},
   }};
```

15 test/test.cpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#include "toy_test.hpp"
#include "toy_compression.hpp"
#include <cstdint>
#include <iomanip>
#include <ios>
#include <iostream>
#include <sstream>
#include <vector>

using container = std::vector<std::uint8_t>;
using iterator  = typename container::iterator;

using test_type        = unsigned;
using signed_test_type = int;

using namespace toy_compression;

using binary_io::make_bit_reader;
using binary_io::make_bit_writer;

/* subsystem test functions... */
#include "binary_io_test.hpp"
#include "elias_delta_test.hpp"
#include "elias_gamma_test.hpp"
#include "golomb_rice_test.hpp"
#include "truncated_binary_test.hpp"
#include "unary_test.hpp"
#include "varint_test.hpp"
#include "zigzag_test.hpp"

int main() {
   toy_test::run_suites(
      {reader_suite, writer_suite, unary_suite, truncated_binary_suite,
       centered_truncated_binary_suite, binary_in_range_suite,
       elias_gamma_suite, elias_delta_suite, golomb_suite, rice_suite,
       zigzag_suite, offset_zigzag_suite, varint_suite} );
}
```

16 test/truncated_binary_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

void test_truncated_binary_coder( test_type value, test_type n ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::truncated_binary::encode( value, n, writer );
   }
   auto reader = make_bit_reader( c );
   auto decoded
      = integer_codes::truncated_binary::decode<test_type>( n, reader );
   ASSERT( decoded == value );
}

toy_test::suite truncated_binary_suite{
   "Test for truncated binary coder",
   {
      {"encode() throws an exception for n = 0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS(
             integer_codes::truncated_binary::encode<test_type>( 0, 0, writer ),
             std::invalid_argument );
       }},
      {"encode() throws an exception for x > n",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS(
             integer_codes::truncated_binary::encode<test_type>( 4, 3, writer ),
             std::invalid_argument );
       }},

      {"decode() throws an exception for n = 0",
       [] {
          container c( 100 );
          auto      reader = make_bit_reader( c );
          THROWS(
             integer_codes::truncated_binary::decode<test_type>( 0, reader ),
             std::invalid_argument );
       }},

      {"encodes a value of 3 using n = 6",
       [] { test_truncated_binary_coder( 3, 6 ); }},
```

```
      {"encodes a value of 3 using n = 4",
       [] { test_truncated_binary_coder( 3, 4 ); }},

      {"encodes a value of 3 using n = 8",
       [] { test_truncated_binary_coder( 3, 8 ); }},

      {"encodes a value of 7 using n = 8",
       [] { test_truncated_binary_coder( 7, 8 ); }},

   }};

void test_centered_truncated_binary_coder( test_type value, test_type n ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::centered_truncated_binary::encode( value, n, writer );
   }
   auto reader  = make_bit_reader( c );
   auto decoded = integer_codes::centered_truncated_binary::decode<test_type>(
      n, reader );
   ASSERT( decoded == value );
}

toy_test::suite centered_truncated_binary_suite{
   "Test for centered truncated binary coder",
   {
      {"encode() throws an exception for n = 0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::centered_truncated_binary::encode<test_type>(
                     1, 0, writer ),
                  std::invalid_argument );
       }},

      {"decode() throws an exception for n = 0",
       [] {
          container c( 100 );
          auto      reader = make_bit_reader( c );
          THROWS( integer_codes::centered_truncated_binary::decode<test_type>(
                     0, reader ),
                  std::invalid_argument );
       }},

      {"encodes a value of 3 using n = 6",
       [] { test_centered_truncated_binary_coder( 3, 6 ); }},

      {"encodes a value of 3 using n = 4",
       [] { test_centered_truncated_binary_coder( 3, 4 ); }},

      {"encodes a value of 3 using n = 8",
       [] { test_centered_truncated_binary_coder( 3, 8 ); }},

      {"encodes a value of 7 using n = 8",
       [] { test_centered_truncated_binary_coder( 7, 8 ); }},
```

```
   }};

void test_binary_in_range( test_type value, test_type low, test_type high ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::binary_in_range::encode( value, low, high, writer );
   }
   auto reader = make_bit_reader( c );
   auto decoded
      = integer_codes::binary_in_range::decode<test_type>( low, high, reader );
   ASSERT( decoded == value );
}

toy_test::suite binary_in_range_suite{
   "Test for binary in range coder",
   {
      {"encode() throws an exception for low = 0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::binary_in_range::encode<test_type>( 1, 0, 10,
                                                                     writer ),
                  std::invalid_argument );
       }},

      {"encode() throws an exception for low > high",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::binary_in_range::encode<test_type>( 1, 4, 1,
                                                                     writer ),
                  std::invalid_argument );
       }},

      {"encode() throws an exception for x < low",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::binary_in_range::encode<test_type>( 3, 4, 5,
                                                                     writer ),
                  std::invalid_argument );
       }},
      {"encode() throws an exception for x > high",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::binary_in_range::encode<test_type>( 6, 4, 5,
                                                                     writer ),
                  std::invalid_argument );
       }},

      {"decode() throws an exception for low = 0",
       [] {
          container c( 100 );
          auto      reader = make_bit_reader( c );
          THROWS(
```

```
            integer_codes::binary_in_range::decode<test_type>( 0, 10, reader ),
            std::invalid_argument );
     }},

    {"decode() throws an exception for low > high",
     [] {
        container c( 100 );
        auto      reader = make_bit_reader( c );
        THROWS(
            integer_codes::binary_in_range::decode<test_type>( 4, 1, reader ),
            std::invalid_argument );
     }},

    {"encodes a value of 3 using range = [2,6]",
     [] { test_binary_in_range( 3, 2, 6 ); }},

    {"encodes a value of 3 using range = [1,4]",
     [] { test_binary_in_range( 3, 1, 4 ); }},

    {"encodes a value of 3 using n = [1, 8]",
     [] { test_binary_in_range( 3, 1, 8 ); }},

    {"encodes a value of 7 using n = [1, 8]",
     [] { test_binary_in_range( 7, 1, 8 ); }},

}};
```

17 test/unary_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

void test_unary_coder( test_type value ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::unary::encode( value, writer );
   }
   auto reader  = make_bit_reader( c );
   auto decoded = integer_codes::unary::decode<test_type>( reader );
   ASSERT( decoded == value );
}

toy_test::suite unary_suite{
   "test for unary coder",
   {
      {"throws an exception for x=0",
       [] {
          container c;
          auto      writer = make_bit_writer( c );
          THROWS( integer_codes::unary::encode<test_type>( 0, writer ),
                  std::invalid_argument );
       }},
      {"encodes a one-value", [] { test_unary_coder( 1 ); }},
      {"encodes a small integer", [] { test_unary_coder( 2 ); }},
      {"encodes another small integer", [] { test_unary_coder( 5 ); }},
   }};
```

18 test/varint_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

void test_varint( test_type value ) {
   container c;
   {
      auto writer = make_bit_writer( c );
      integer_codes::varint::encode( value, writer );
   }
   auto reader  = make_bit_reader( c );
   auto decoded = integer_codes::varint::decode<test_type>( reader );
   ASSERT( decoded == value );
}

/* suites */
toy_test::suite varint_suite{"Test␣for␣varint␣coder",
                             {
                                {"encodes␣0", [] { test_varint( 0 ); }},
                                {"encodes␣1", [] { test_varint( 1 ); }},
                                {"encodes␣128", [] { test_varint( 128 ); }},
                                {"encodes␣275", [] { test_varint( 275 ); }},
                                {"encodes␣1,948", [] { test_varint( 1948 ); }},
                                {"encodes␣65538", [] { test_varint( 65538 ); }},
                             }};
```

19 test/zigzag_test.hpp

```
/*
 * Toy Compression - Toy Compression Code
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

void test_zigzag( signed_test_type value ) {
   auto encoded = integer_codes::zigzag::encode( value );
   auto decoded = integer_codes::zigzag::decode( encoded );
   ASSERT( decoded == value );
}

void test_offset_zigzag( signed_test_type value, signed_test_type offset ) {
   auto encoded = integer_codes::offset_zigzag::encode( value, offset );
   auto decoded = integer_codes::offset_zigzag::decode( encoded, offset );
   ASSERT( decoded == value );
}

toy_test::suite zigzag_suite{"Test for zigzag coder",
                             {
                                {"encodes 0", [] { test_zigzag( 0 ); }},
                                {"encodes 1", [] { test_zigzag( 1 ); }},
                                {"encodes -1", [] { test_zigzag( -1 ); }},

                             }};

toy_test::suite offset_zigzag_suite{
   "Test for offset zigzag coder",
   {
      {"encodes 0 with offset 12", [] { test_offset_zigzag( 0, 12 ); }},
      {"encodes 1 with offset 12", [] { test_offset_zigzag( 1, 12 ); }},
      {"encodes -1 with offset 12", [] { test_offset_zigzag( -1, 12 ); }},

   }};
```

20 toy_test/toy_test.hpp

```
/*
 * Toy Test - Toy Unit Testing
 * Written in 2018 by Gerald Lewis <lewisgdljr@gmail.com>
 *
 * To the extent possible under law, the author(s) have dedicated all copyright
 * and related and neighboring rights to this software to the public domain
 * worldwide. This software is distributed without any warranty.
 * You should have received a copy of the CC0 Public Domain Dedication along
 * with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 */

#pragma once
#ifndef TOY_TEST_HPP_INCLUDED
#define TOY_TEST_HPP_INCLUDED

#include <functional>
#include <initializer_list>
#include <iostream>
#include <vector>

namespace toy_test {
   struct test_case {
      const char*           name;
      std::function<void()> run;
      void                  operator()() const { run(); }
   };

   struct failure {
      const char* expr;
      int         line;
   };

   struct suite {
      const char*            name;
      std::vector<test_case> tests;

      bool run() const {
         bool ok{true};
         std::cout << "[SUITE] Running test suite: \"" << name << "\""
                   << std::endl
                   << std::endl;
         for ( auto&& test : tests ) {
            try {
               test();
               std::cout << "[OK.] \"" << test.name << "\" passed."
                         << std::endl;
            } catch ( failure& caught ) {
               ok = false;
               std::cout << "[FAIL!] \"" << test.name << "\" failed."
                         << std::endl;
               std::cout << "Failing condition: \"" << caught.expr
                         << "\" at line: " << caught.line << std::endl;
            }
         }
```

```
        if ( ok ) {
            std::cout << std::endl
                      << "[OK] All tests passed for suite: \"" << name << "\""
                      << std::endl;
        } else {
            std::cout << std::endl
                      << "[FAIL!] Test failures detected in suite: \"" << name
                      << "\"" << std::endl;
        }
        return ok;
    }
};

bool run_suite( suite const& suite ) { return suite.run(); }

bool run_suites( std::initializer_list<suite const> const& suites ) {
    bool ok = true;
    for ( auto const& a : suites ) {
        ok &= run_suite( std::forward<suite const>( a ) );
    }

    if ( ok ) {
        std::cout << std::endl
                  << "[OK] All tests passed." << std::endl
                  << std::endl;
    } else {
        std::cout << std::endl
                  << "[FAIL!] Test failures detected." << std::endl
                  << " Check the output for details." << std::endl
                  << std::endl;
    }
    return ok;
}

#define ASSERT( condition )                          \
    void( ( condition ) ? 0                          \
                        : throw toy_test::failure( \
                              {"ASSERT(" #condition ")", __LINE__} ) )

#define THROWS( expression, exception )                                  \
    try {                                                                \
        ( expression );                                                  \
        throw toy_test::failure(                                         \
            {"THROWS(" #expression ", " #exception ")", __LINE__} ); \
    } catch ( exception& ) {                                             \
    } catch ( ... ) {                                                    \
        throw toy_test::failure(                                         \
            {"THROWS(" #expression ", " #exception ")", __LINE__} ); \
    }
} // namespace toy_test
#endif // TOY_TEST_HPP_INCLUDED
```