Contents

1 demo.cpp

```cpp
/*
 * 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/>.
 */

#include "factorial.hpp"
#include "leap_year.hpp"
#include "testing.hpp"

toy_test::suite leap_year_suite{
   "Test␣for␣leap␣year␣formula",
   {{"odd␣years␣are␣not␣leap␣years", [] { ASSERT( !is_leap_year( 2001 ) ); }},

    {"even␣years␣which␣are␣not␣multiples␣of␣4␣are␣not␣leap␣years",
     [] { ASSERT( !is_leap_year( 2002 ) ); }},

    {"multiples␣of␣4␣but␣not␣100␣are␣leap␣years",
     [] { ASSERT( is_leap_year( 1996 ) ); }},

    {"multiples␣of␣100␣but␣not␣400␣are␣not␣leap␣years",
     [] { ASSERT( !is_leap_year( 1900 ) ); }},

    {"multiples␣of␣400␣are␣leap␣years", [] { ASSERT( is_leap_year( 2000 ) ); }},

    {"years␣before␣1752␣are␣not␣valid",
     [] { THROWS( is_leap_year( 800 ), std::exception ); }}}};

toy_test::suite factorial_suite{
   "Test␣for␣factorial",
   {
      {"0!␣==␣1", [] { ASSERT( factorial( 0 ) == 1 ); }},
      {"3!␣==␣6", [] { ASSERT( factorial( 3 ) == 6 ); }},
      {"10!␣==␣3628800", [] { ASSERT( factorial( 10 ) == 3628800 ); }},
   }};

int main() { toy_test::run_suites( {leap_year_suite, factorial_suite} ); }
```

2 factorial.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/>.
 */

int factorial( int n ) {
   if ( n < 1 )
      return 1;
   return n * factorial( n - 1 );
}
```

3 leap_year.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/>.
 */

#define INTENTIONAL_FAILURE
bool is_leap_year( int year ) {
#ifndef INTENTIONAL_FAILURE
   if ( year < 1752 ) {
      // is the year one in which the Gregorian calendar
      // was used in the British Empire and/or USA?
      throw std::invalid_argument(
         "The␣Gregorian␣calendar␣wasn't␣used␣in␣the␣"
         "British␣Empire␣(and␣therefore␣the␣American␣colonies)␣before␣1752!" );
   }
#endif // INTENTIONAL_FAILURE

   if ( ( year & 3 ) || ( !( year % 100 ) && ( year % 400 ) ) ) {
      // is the year odd or not a multiple of 4?
      // or is the year an even century but NOT a multiple of 400 years?
      return false;
   }

   return true;
}
```

4 testing.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 TESTING_HPP_INCLUDED
#define TESTING_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 // TESTING_HPP_INCLUDED
```