CS3520 Programming in C++
Testing

Ken Baclawski
Fall 2016

Outline

- Boost Testing
- Assignment #8

Using Boost Unit Test

- Install the Boost Unit Test library
- Add the following to your compile command:

```
-lboost_unit_test_framework
```

- The `Main.cpp` file will be changed
- No other file is changed
- Compile and run as usual, except for the compile option shown above

Test program
for the shared2 package

Unit Test Structure

- Tests are divided into *modules*
  - We will develop one test module for each assignment
- Modules are divided into *test cases*
- Test cases consist of code intermixed with *checks*
  - Behavior depends on the severity level of the check
- Exceptions are caught and reported
  - Terminates the test case but not the module
  - No try-catch blocks are needed

Severity Levels

- WARN gives a warning
  - Only shown when `--log_level=warning` is on the command line of the test program
  - Does not affect whether the test program succeeds
- CHECK gives an error but continues the test case
  - Included in the failure count
- REQUIRE gives a fatal error and terminates the test case
  - The test module continues
  - Included in failure count
- If there any failures then the test program returns with value 201

Using the Boost Unit Test Library

- The Boost Unit Test library can be used in many ways
  - The simplest is to modify your `Main.cpp` file
  - The other files are not modified
- The following slides show how the original `Main.cpp` file for the `shared2` package was modified
  - Additional test cases were added
  - Original code was included as one test case

Main.cpp

This might be different for your installation of Boost

Title of your test module

```
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Test of the shared4 package
#include <boost/test/unit_test.hpp>

#include <string>
#include <iostream>
#include <stdexcept>
#include <thread>
#include <chrono>

#include "SharedStack.h"
#include "Action.h"
```

Boost unit test library

This is the same as the original Main.cpp file

Main.cpp continued

```
...
#include "SharedStack.h"
#include "Action.h"

using namespace std;
using namespace shared4;

/**
 * @namespace shared4 This package illustrates the
 * producer-consumer design pattern with a bounded
 * container.  Testing is performed using Boost.
 *
 * @author Ken Baclawski
 */
...
```

This is the same as the original Main.cpp file

Main.cpp continued: Test Case 1

```
...
BOOST_AUTO_TEST_CASE(shared_stack_test1) {
  SharedStack sharedStack(2);
  sharedStack.produce("object");
  sharedStack.produce("object");
  BOOST_REQUIRE_EQUAL(sharedStack.consume(), "wrong");
  BOOST_CHECK_EQUAL(sharedStack.consume(), "wrong");
}
...
```

The first check fails so the second check is not attempted

Main.cpp continued: Test Case 2

```
...
BOOST_AUTO_TEST_CASE(shared_stack_test2) {
  SharedStack sharedStack(10);
  sharedStack.produce("object");
  sharedStack.produce("object");
  BOOST_CHECK(sharedStack.consume() == "wrong");
  BOOST_WARN_EQUAL(sharedStack.consume(), "wrong");
}
...
```

The first check fails but it is not required so the second check will be attempted

The second check fails but it is only a warning so it will be shown only if it is requested

Main.cpp continued: Test Case 3

```
BOOST_AUTO_TEST_CASE(shared_stack_test3) {

  // Construct the bounded shared stack.

  SharedStack sharedStack(1);

  // Construct two action objects.

  Action produce("produce", sharedStack);
  Action consume("consume", sharedStack);
...
  // Wait until the threads have finished.

  t1.join();
  t2.join();
  t3.join();
  t4.join();
}
```

This is the same as the original Main.cpp

The try and catch were not included because Boost will catch any exceptions and report on them

Output

```
Running 3 test cases...
An item has been produced and there are now 1 item(s)
An item has been produced and there are now 2 item(s)
An item has been consumed and there are now 1 item(s)
src/shared4/Main.cpp(29): fatal error in "shared_stack_test1":
  critical check sharedStack.consume() == "wrong" failed [object != wrong]
An item has been produced and there are now 1 item(s)
An item has been produced and there are now 2 item(s)
An item has been consumed and there are now 1 item(s)
src/shared4/Main.cpp(37): error in "shared_stack_test2":
  check sharedStack.consume() == "wrong" failed
An item has been consumed and there are now 0 item(s)
An item has been produced and there are now 1 item(s)
An item has been consumed and there are now 0 item(s)
Going to sleep for a while...
An item has been produced and there are now 1 item(s)
An item has been consumed and there are now 0 item(s)

*** 2 failures detected in test suite "Test of the shared4 package"
```

Basic Boost Macros

- `BOOST_<level>(`predicate`)`
  - The predicate is evaluated and if false then the check fails and is reported according to the severity level
- `BOOST_<level>_MESSAGE(`predicate, message`)`
  - Uses a custom message
- `BOOST_<level>_PREDICATE(`function, (arg1)(arg2)...`)`
  - Evaluates the function with the specified arguments
  - Note the unusual syntax
  - Has the advantage of reporting the argument values

Boost Comparison Macros

```
BOOST_<level>_BITWISE_EQUAL
BOOST_<level>_EQUAL
BOOST_<level>_EQUAL_COLLECTIONS
BOOST_<level>_GE
BOOST_<level>_GT
BOOST_<level>_LE
BOOST_<level>_LT
BOOST_<level>_NE
```

- The two arguments are separately evaluated and then compared
- Only use these for relatively simple comparisons
- For more complex conditions use the basic macros

Floating-Point Comparison Macros

```
BOOST_<level>_CLOSE
BOOST_<level>_CLOSE_FRACTION
BOOST_<level>_SMALL
```

- The first two arguments are separately evaluated and then compared
- The third argument specifies how close the arguments must be
  - Percentage
  - Fraction
- For `SMALL` the comparison is with 0.0

Testing for Exceptions

- `BOOST_<level>_THROW`(statement, exception)
  - Executes the statement
  - If no exception is thrown, then the check fails
  - If an exception is thrown, it must have the type specified in the second argument
- `BOOST_<level>_EXCEPTION`(statement, exception, predicate)
  - Same as above but also evaluates a predicate
  - Useful for checking that the thrown exception is the correct one
- `BOOST_<level>_NO_THROW`(statement)
  - Executes the statement
  - If an exception is thrown, then the check fails

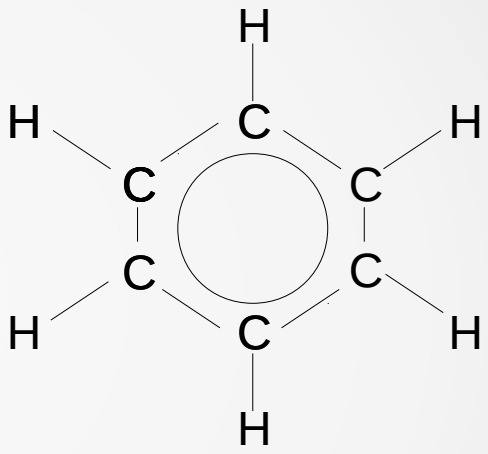
Assignment #8

Requirements

- As in Assignment #7, develop two classes: `Atom` and `Molecule`
- A molecule has a name and contains a set of bonds
- A bond is a set of atoms
  - A molecule contains a set of sets of atoms
- An atom has a symbol and is part of a molecule
- The differences between this assignment and #7 will be highlighted

Example Molecule

- The name of this molecule is benzene
- It has twelve atoms and seven bonds
- Six of the bonds are between C and H atoms
- The formula is C6H6
- The letters C and H are the symbols of the atoms
- The set of all six C's form one bond



Atom Class

- This class is the same as the Atom class in Assignment #7 except for the namespace

Molecule Class

- This class is the same as the Molecule class in Assignment #7 except as follows
- Add a method `traverse` has a function parameter
  - The function has two parameters
    - `id` identifies which bond is being traversed
    - `atom` is a pointer to the current atom being traversed
  - The function is applied to every atom in every bond
- Use the `traverse` method to compute the `getAtoms` method
- Add a method `bondCount` that counts the number of bonds that contain a specified atom

Main Program

- First Boost Test Case
  - Construct water, salt (NaCl), and the empty molecule
  1. Check that the molecules have the correct names
  2. Check that the molecules have the correct formulas
  3. Check that each molecules has the correct set of atoms
  4. Check that every atom belongs to the correct molecule
  5. Check that every atom has the correct bond count
  - Comment each check with what is being checked
- Second Boost Test Case
  - Check that an exception is thrown if an atom is in two molecules
- Also validate memory management with `valgrind` or equivalent

Next Class

- Generic Programming