CS 6015: Software Engineering

Spring 2024

Lecture 4: Testing – Testing I/O

This Week

- Version/Source Control
- Testing – I/O Testing
- Assignment 2 released - you can complete the testing part after this lecture

Next Week

- Quiz 1: Tuesday January 23, 2024 (~ 10 minutes)
- Designing a program
- Debugging

Plan

- Motivation
- Testing principles
- Testing in Java
- Testing in C++
  - Catch 2
- Testing programs with I/O
- DEMO

Software testing

- Running the code on carefully chosen input and checking the results.
- Objective:
  - Finding software bugs
- Validating and verifying that the software/application meets the requirements for selected input

Why you think testing is NOT important?

- Need to finish fast – testing will slow me down
- Programming many years – my code is perfect – Don’t insult me
- Using built-in data structures in my code – they work perfectly – no need to test my code!

Problems caused by insufficient testing

April 1994: Airplane crash - 264 People dead

The aircraft had not received the update at the time of the crash because XXX Airlines judged that the modifications were not urgent

May 1996: Largest error in US banking

Software "glitches" caused the bank accounts of 823 customers to be credited with $924 million each.

Testing is important

- Testing to ensure correctness / helps with design
- Software bugs can be expensive
- Dangerous when it comes to human lives
- Strategy for testing?
  - Objective while testing is to make the program fail
  - Good testing beats the program everywhere it might be vulnerable

Testing principles

- Systematic
  - Arbitrary testing: likely to miss bugs
  - Exhaustive testing: often impossible
- Early and often
  - Late testing would lead to painful debugging at later stages
  - Testing for each method/function/class
  - Confident when making changes
- Automatic
  - Eliminate user input while testing

Systematic testing

- Divide the large input space into a few representatives
- Pick a set of test cases for each representative
- Small enough to run quickly, yet large enough to validate the program
- Remember exhaustive test cases
  - Often impossible
  - Requires testing resources and time

Systematic testing: example

```
// return the index of x in the sorted array, -1 if not present
int binarySearch(int arr[], int low, int high, int x)
{
    if (high > low)
    {
        int mid = low + (high - low) / 2;

        if (arr[mid] == x)
            return mid;

        if (arr[mid] > x)
            return binarySearch(arr, low, mid - 1, x);

        return binarySearch(arr, mid + 1, high, x);
    }
    return -1;
}

// Note that the above code is buggy
```

Representative inputs for array parameters

- Generally, when testing a function with lists!

Representative inputs for array parameters

- Generally, when testing a function with lists!
  - Include boundaries
  - Empty lists
  - Lists with 1 element
  - Unique lists
  - Duplicate elements in the list
  - Sorted and unsorted lists
  - Odd number of elements
  - Even number of elements

Representative inputs for string parameters

- Testing if a string is palindrome

Representative inputs for string parameters

- Generally, when testing a function with strings
  - Empty strings
  - 1 char in string
  - String with even/odd number of elements
  - Strings with lower/upper case elements

Representative inputs for file parameters

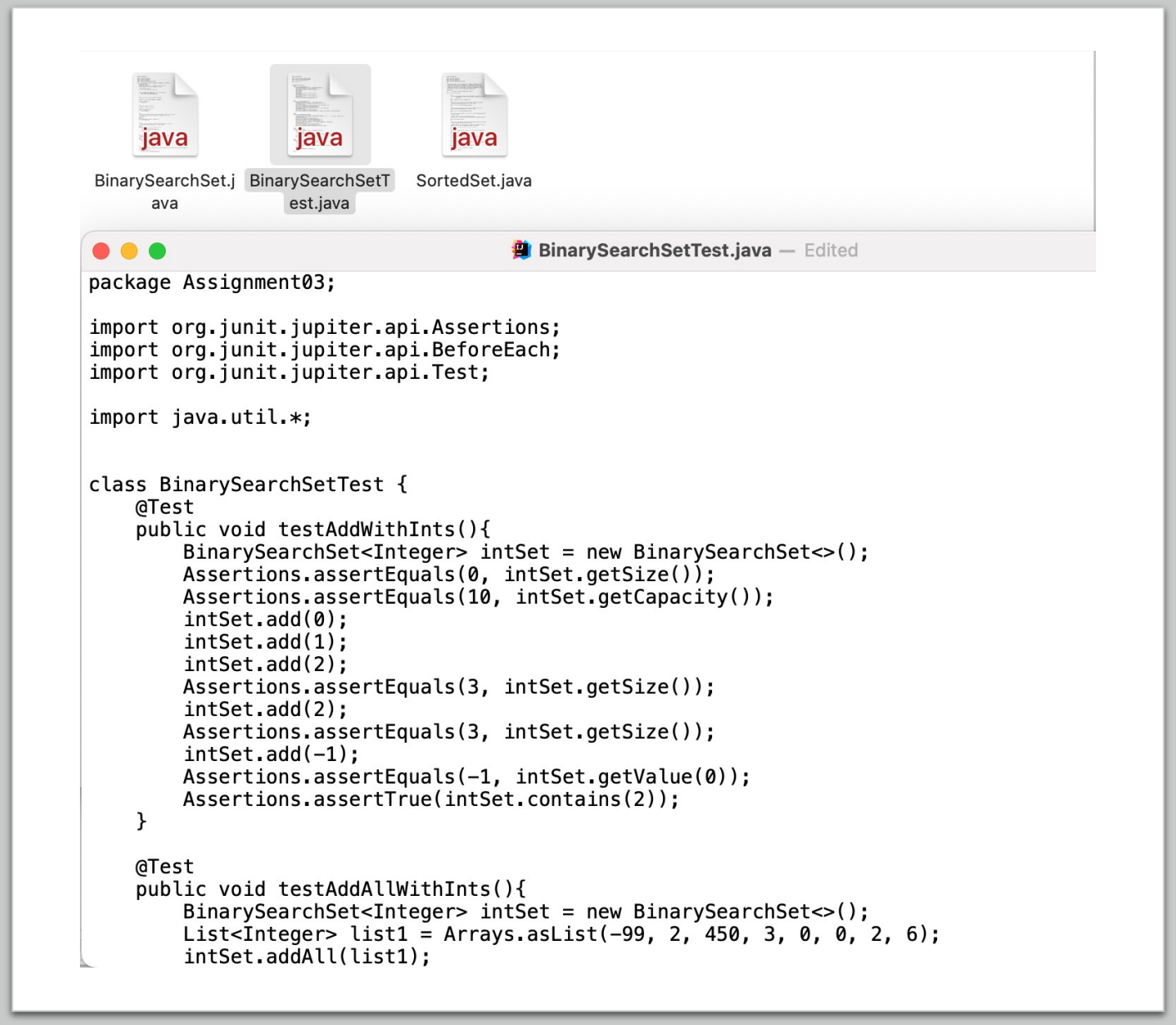
- Pacman

Representative inputs for file parameters

- Generally, when testing a function with files
  - Empty files
  - File does not exist
  - Files with small/medium/large sizes

Testing frameworks

- Java
  - Junit
- C++
  - CPP Unit
  - Google test
  - Catch2
  - ….
- Common definitions
  - Test case: refers to a group of related checks
  - Test suite: collection of test cases



Testing frameworks: CPPUnit

```
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>
#include <cppunit/BriefTestProgressListener.h>

int main()
{
    CPPUNIT_NS::TestResult tr;
    CPPUNIT_NS::TestResultCollector  tr_ctrl;
    tr.addListener(&tr_ctrl);
    CPPUNIT_NS::BriefTestProgressListener btp_lstnr;
    tr.addListener(&btp_lstnr);

    CPPUNIT_NS::TestRunner runner;
    runner.addTest(CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest());
    runner.run(tr);

    CPPUNIT_NS::CompilerOutputter compileroutputter(&tr_ctrl, std::cerr);
    compileroutputter.write();


    return 0;
}
```

Testmain.cpp example using cppunit

Testing frameworks: CPPUnit

```
#include <cppunit/TestFixture.h>
//include the TestFixture header file
#include <cppunit/extensions/HelperMacros.h>
#include "xxx.h"//header file of class to be tested
class xxxtest: public CPPUNIT_NS::TestFixture {

public:
void setUp();
void tearDown();

protected:
//Declare testcases
CPPUNIT_TEST (getmaxTest);
CPPUNIT_TEST (getminTest);

private:
//declare instances to be used in the
};
```

xxxtest.h example using cppunit

Testing frameworks: CPPUnit

```
#include "xxxtest.h"
CPPUNIT_TEST_SUITE_REGISTRATION(xxtest);
void listTest::setUp(){
  int A[3]={1,2,3};
  a = new List (A, 3);

  int F[4]={1,4,0, 0};
  f = new List (F, 4);
}

void listTest::tearDown(){
  delete a, f;
}

void listTest :: getmaxTest (){
  CPPUNIT_ASSERT_EQUAL_MESSAGE("Max at the end", a->getMax(), 3);
  CPPUNIT_ASSERT_EQUAL_MESSAGE("Max in the middle", d->getMax(), 4);}

void listTest :: getminTest ()
{
  CPPUNIT_ASSERT_EQUAL_MESSAGE("Min at the end",d->getMin(), 0);
}
```

xxxtest.cpp example using cppunit

Testing frameworks: Catch2

- Catch 2: testing framework that we will use
  - Minimal steps to perform testing
  - Assertions look normal c++ Booleans
  - Include the header (1 header) and write the test cases

- Reference: **https://github.com/catchorg/Catch2/blob/v2.x/docs/tutorial.md**

Catch2: writing tests

binarysearch.cpp

```
#include "catch.h"

int binarySearch(int arr[], int low, int high, int x)
{
   //...
}

TEST_CASE( "Search Element in Array")
{
        int a[8] = {4, 9, 21, 33, 35, 50, 55, 60};

        CHECK ( binarySearch (a,0,7,35) == 4 );
        CHECK ( binarySearch (a,0,7,33) == 3 );
        CHECK ( binarySearch (a,0,7,4) == 0 );
        CHECK ( binarySearch (a,0,7,60) == 7 );

        CHECK ( binarySearch (a,0,7,3) == -1 );
        CHECK ( binarySearch (a,0,7,30) == -1 );
}
```

main.cpp

```
#define CATCH_CONFIG_RUNNER //before include
#include "catch.h"
int main(int argc, char **argv) {
  Catch::Session().run(argc, argv);
  return 0;
}
```

Catch2: CHECK

- **CHECK**: tests an expression and continues even if the assertion fails.

- Is a Macro: look like functions but they are slightly different. Catch uses macros for nice error reporting.

- Other assertion macros:
  - **CHECK_FALSE**(expression): asserts that expression evaluates to false
  - **REQUIRE**: tests an expression and aborts if it fails.
  - **REQUIRE_FALSE**(expression): No added value to use for plain bool variable.
  - **CHECK_THROWS**(expression): ensure that expression throws an exception

Catch2: macros

- Macros
  - Like functions but not functions

```
#include <iostream>
#define min(a, b) (((a) < (b)) ? (a) : (b)) //Macros definition
int main()
{
    int x = 10;
    int y = 12;
    std::cout << "Min: " << min(x, y);
    return 0;
}
```

(cond ? a : b) has the value of a if cond is true, otherwise b


```
//preprocessing replaces min (x, y) by
(((x) < (y)) ? (x) : (y))
```

Macros in catch2.h

```
#define REQUIRE( ... )
```

Catch2: Sections

```
#include "catch.h”
int binarySearch(int arr[], int low, int high, int x)

TEST_CASE( "Search_Element_in_Array")
{
    int a[8] = {4, 9, 21, 33, 35, 50, 55, 60};
    SECTION("Element_anywhere_middle_array")
    {
        REQUIRE( binarySearch (a,0,7,35) == 4 );
    }
    SECTION(”edge_cases")
    {
        REQUIRE( binarySearch (a,0,7,4) == 0 );
        REQUIRE( binarySearch (a,0,7,60) == 7 );
    }
    SECTION("Element_not_in_array")
    {
        REQUIRE( binarySearch (a,0,7,3) == -1 );
    }
}
```

```
> c++ binarysearch.cpp main.cpp -std=c++11  -o binary
> ./binary -c Element_not_in_array
> ./binary -c edge_cases
```

Can run particular test case by adding tags

```
> ./binary Search_Element_in_Array
```

Catch2: Sections

```
#include "catch.h"

TEST_CASE("increment_x")
{
    int x=10; //like the junit setup() method
    SECTION("inc1"){
        x++;
        CHECK( x==11 );
    }

    SECTION("inc2"){
        CHECK( x==10 );
    }
}
```

Catch2: Sections

```
#include "catch.h"

TEST_CASE("increment_x")
{
    int x=10; //like the junit setup() method
    SECTION("inc1"){
        x++;
        CHECK( x==11 );
    }

    SECTION("inc2"){
        //x is 10 at the start of each section
        CHECK( x==10 );
    }
}
```

Compile without testcase

How to compile without testcases?

- **Save the testcases in separate cpp file**
- **Compile without including the test cases file**

Catch2: Revisiting writing tests for the project

cmdline.cpp

```
#define CATCH_CONFIG_RUNNER
#include "catch.h”


Catch::Session().run(1, argv); // should be called when argv[1]==“--test”
```

Testing programs with I/O

Programs with I/O

- How to test programs with input/output?
- Consider the example

```
#include <iostream>
#include <string>

static void say_hello() {
std::string name;
std::cin >> name;
std::cout << "Hello, " << name << "!\n"; }

int main(int argc, char **argv) {
say_hello(); }
```

No parameters – no return value

Can we use: **CHECK** (say_hello() == "Hello, XYZ!");

Solution: Add parameters to accept I/O

Testing programs with I/O

- Approach 1: Pass the string as parameter then test.

```
#include <iostream>
#include <string>

static std::string say_hello(std::string name) {  // now we have function that has args
                                                   // and returns a value. Test as before
  return "Hello, " + name + "!\n";
}

int main(int argc, char **argv) {

   std::cout << say_hello(std::string name);      // to reflect the new changes
}
```

Now, we can use: **CHECK** (say_hello("XYZ") == "Hello, XYZ!");

Testing programs with I/O

- Example

```
static std::string say_hello(std::string name) {
  return "Hello, " + name + "!\n";
}

TEST_CASE( "hello" ) {
  CHECK( say_hello(”Lynn") == "Hello, Lynn!\n" );
  CHECK( say_hello(”Roy") == "Hello, Roy!\n" );
  CHECK( say_hello("world") == "Hello, world!\n" );
}
```

Testing programs with I/O

Another approach by using I/O streams

- Step 1: Pass the I/O streams as parameters

```
#include <iostream>
#include <string>
static void say_hello(std::istream &in, std::ostream &out) {

  std::string name;
  in >> name;
  out << "Hello, " << name << "!\n";
}

int main(int argc, char **argv) {
  say_hello(std::cin, std::cout);    //effectively &std::cin
}
```

```
& is like * but without changing:
1-  . to ->
2-  in << ... to (*in) << ...
```

Testing programs with I/O

- String streams: associates a string object with a stream allowing to read from the string as if it was a stream
- Objects of this class use a string buffer that contains a sequence of characters.
- Can be accessed directly as a string object, using member str.

Testing programs with I/O

- Step 2: Create string streams

```
TEST_CASE( "hello" ) {
  std::stringstream in(”Lynn"); // string stream: is a specific istream object
  std::stringstream out(""); // string stream
  say_hello(in, out);
  CHECK( out.str() == "Hello, Lynn!\n" );    // To extract an output string
}
```

Testing programs with I/O

- How to reduce repeated code in tests?

```
TEST_CASE( "hello" ) {
    {
        std::stringstream in(”Lynn");
        std::stringstream out("");
        say_hello(in, out);
        CHECK( out.str() == "Hello, Lynn!\n" );
    }

    {
        std::stringstream in(”Roy");
        std::stringstream out("");
        say_hello(in, out);
        CHECK( out.str() == "Hello, Roy!\n" );
    }
}
```

Testing programs with I/O

- Solution: test helper method

```
static std::string say_hello_string(std::string s) {
  std::stringstream in(s);
  std::stringstream out("");
  say_hello(in, out);
  return out.str();
}

TEST_CASE( "hello" ) {
  CHECK( say_hello_string("Dolly") == "Hello, Dolly!\n" );
  CHECK( say_hello_string("Kitty") == "Hello, Kitty!\n" );
  CHECK( say_hello_string("world") == "Hello, world!\n" );
}
```