Testing

About

It’s quite possible that you haven’t put a lot of thought into testing up to this point.

Generally, students test their programs by running it multiple times and trying out permutations of inputs. If this is what you do, you’ve probably found it tedious after a while.

But there’s more to testing than just running the program a bunch of times manually…

Topics

1. Why test?
2. Types of testing
3. Unit tests
4. D

Why Test?

Jumping right in

Many students begin developing their school projects by *jumping right into the code*. Most of the time, no time is spent on *design* or on planning how to *test* the project. Often, very little time is even spent on the *specification* itself!

Notes

1. Why Test?

Software Development is more than programming

The field of programming is more about just hitting keys in a text editor and writing code. Software Engineering is about figuring out requirements, designing the solution, and being able to validate the work done.

Notes

1. Why Test?

Manual testing is tedious

If you've spent time testing a program by running it multiple times and trying different inputs, you've probably found it pretty tedious. But it is important to check all reasonable inputs and outputs to ensure your program works.

It doesn't all have to be done *manually*, either. If you design your program to be tested, you can write functions that test your program automatically.

Notes

1. Why Test?

What is the goal?

Coming up with an idea of tests *prior* to starting the project will help you **explicitly** state what the program should do.

It will also give you a list of tasks that need to be handled, without having to try to figure out what (and how) to test after-the-fact.

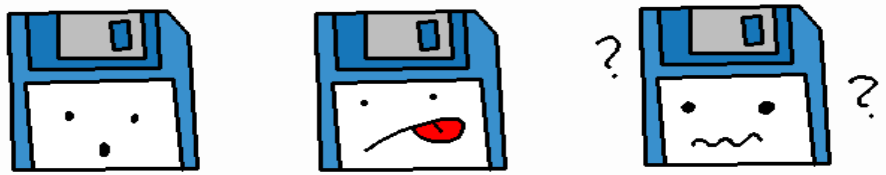
Notes

1. Why Test?

How do you validate others' code?

If you're working on a team, how do you ensure that the software works? How do you make sure that *someone else* doesn't break the codebase?

Having test sets will help you validate your own and others' work.



Notes

Types of Testing

2. Types of Tests

Integration tests: is the phase in software testing in which individual software modules are combined and tested as a group.

(From Wikipedia https://en.wikipedia.org/wiki/Integration_testing)

Unit tests: a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

(From Wikipedia https://en.wikipedia.org/wiki/Unit_testing)

Regression tests: a type of software testing which verifies that software which was previously developed and tested still performs the same way after it was changed or interfaced with other software.

(From Wikipedia https://en.wikipedia.org/wiki/Regression_testing)

Notes

2. Types of Tests

Unit tests vs. Integration tests:

- http://ilkinulas.github.io/assets/integration_tests/no_integration_test.gif
- http://pngif.com/i/7/e/7eKzmepG9WucihJASweHGD.gif
- https://natooktesting.files.wordpress.com/2017/08/unittest_faucet.gif
- https://twitter.com/D_R_French/status/851187144584572928
- https://twitter.com/Dadash91/status/831317037037285376

Notes

2. Types of Tests

In this class, you will be able to validate your homework projects with unit tests written by the instructor, but it is also important to know what a unit test is and how to write one yourself.

Notes

2. Types of Tests

Different companies have different policies on testing; some companies don't maintain unit tests (ugh) and some do.

You'll probably have a better time at a place that maintains their unit tests, rather than takes the attitude,
"Tests? That's just more code to maintain!"

Notes

Unit Tests

3. Unit Tests

Unit Tests are meant to test small units of the program. This generally means the functions.

Functions can be broken down into a simple “these things go in, this thing comes out” way of working, and if your functions are built to be testable, you can write a program to test your functions automatically for you.

Notes

3. Unit Tests

For a function that takes in some inputs and returns some output, you can easily make a list of test cases with expected outputs…

Function: `bool CanDrinkBeer( int age )`

test	input: age	expected output	actual output
1	5	false	
2	15	false	
3	21	true	
4	30	true	

Notes

3. Unit Tests

After you have test cases, you run the function with the inputs and check if the actual output matches the expected output.

test	input: age	expected output	actual output
1	5	false	**true**
2	15	false	false
3	21	true	true
4	30	true	**false**


```
bool CanDrinkBeer( int age )
```

Notes

3. Unit Tests

If there is a mismatch, you can look in the function call and step through the code to find the error.

test	input: age	expected output	actual output
1	5	false	**true**
2	15	false	false
3	21	true	true
4	30	true	**false**


```
bool CanDrinkBeer( int age )
```

Notes

3. Unit Tests

Common error: Don't change your tests to pass! You should be fixing the functionality, not "fixing" the tests!

test	input: age	expected output	actual output
1	5	~~false~~ **true**	**true**
2	15	false	false
3	21	true	true
4	30	~~true~~ **false**	**false**


```
bool CanDrinkBeer( int age )
```

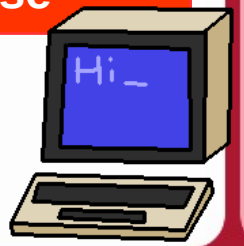
Notes

3. Unit Tests

Common error: Don't change your tests to pass! You should be fixing the functionality, not "fixing" the tests!

test	input: age	expected o	actual output
1	5	~~false~~ **true**	**true**
2	15	false	false
3	21	true	true
4	30	~~true~~ **false**	**false**

No!




```
bool CanDrinkBeer( int age )
```

Notes

3. Unit Tests

Project 3 from last semester running the unit tests

```
        * Insert A-B-C and validate that the root's right child is B          ... pass
        * Insert A-B-C and validate that the B has NO left child              ... pass
        * Insert A-B-C and validate that the B's right child is C             ... pass
        * Insert A-B-C and validate that the size is 3                        ... pass
        * Insert C-B-A and validate that the root exists                      ... pass
        * Insert C-B-A and validate that the root is C                        ... pass
        * Insert C-B-A and validate that the root has NO right child          ... pass
        * Insert C-B-A and validate that the root's left child is B           ... pass
        * Insert C-B-A and validate that the B has NO right child             ... pass
        * Insert C-B-A and validate that the B's left child is A              ... pass
        * Insert C-B-A and validate that the size is 3                        ... pass

Running testset 2 out of 11:    Contains()
        * Tree size 1, item contained in list returns true                   ... pass
        * Tree size 4, item contained in list returns true                   ... pass
        * Tree size 1, item NOT contained in list returns false              ... pass
        * Tree size 4, item NOT contained in list returns false              ... pass

Running testset 3 out of 11:    FindNode()
        * Tree size 1, find item in tree, return not null                    ... pass
        * Tree size 4, find item in tree, return not null                    ... pass
        * Tree size 4, find item in tree, return correct key                 ... pass
        * Tree size 4, find item NOT in tree, return null                    ... pass

Running testset 4 out of 11:    FindParentOfNode()
        * Try to find parent of root; return null                            ... pass
        * Try to find parent of non-root; return parent is not null          ... pass
        * Try to find parent of non-root; return parent is correct key       ... pass
        * Try to find parent of item not in tree; return null                ... pass
```

3. Unit Tests

Project 3 from last semester, unit test report

Test_Insert	Insert C-B-A and validate that the B has NO right child	Insert	passed	B's right child = nullptr	B's right child = 1	
Test_Insert	Insert C-B-A and validate that the B's left child is A	Insert	passed	B's left child = A	B's right child = A	
Test_Insert	Insert C-B-A and validate that the size is 3	Insert	passed	m_nodeCount = 3	m_nodeCount = 3	
SUMMARY FOR Test_Insert: 22 out of 22 tests passed						
Test_Contains	Tree size 1, item contained in list returns true	Insert, Contains	passed	found = 1	found = 1	
Test_Contains	Tree size 4, item contained in list returns true	Insert, Contains	passed	found = 1	found = 1	
Test_Contains	Tree size 1, item NOT contained in list returns false	Insert, Contains	passed	found = 0	found = 0	
Test_Contains	Tree size 4, item NOT contained in list returns false	Insert, Contains	passed	found = 0	found = 0	
SUMMARY FOR Test_Contains: 4 out of 4 tests passed						

3. Unit Tests

Project 3 from last semester, unit test report

Test_Insert	Insert C-B-A and validate that the B has NO right child	Insert	passed	B's right child = nullptr	B's right child = 1	
Test_Insert	Insert C-B-A and validate that the B's left child is A	Insert	passed	B's left child = A	B's right child = A	
Test_Insert	Insert C-B-A and validate that the size is 3	Insert	passed	m_nodeCount = 3	m_nodeCount = 3	
SUMMARY FOR Test_Insert: 22 out of 22 tests passed						
Test_Contains	Tree size 1, item contained in list returns true					
Test_Contains	Tree size 4, item contained in list returns true					
Test_Contains	Tree size 1, item NOT contained in list returns false					
Test_Contains	Tree size 4, item NOT contained in list returns false	Insert, Contains	passed	found = 0	found = 0	
SUMMARY FOR Test_Contains: 4 out of 4 tests passed						

Depending on how you write your unit tests, you can add messages that will give you more information on what is passing or failing.

This Binary Search Tree's Insert test function has 22 tests to check different scenarios when inserting new data, including checking if the size is incremented correctly, and if the node's

3. Unit Tests

When a test fails, if you add good descriptions, you can more easily figure out what went wrong.

Test set	Test	Prerequisite functions Functions that need to be implemented for these tests to work right	Pass/fail	Expected output The output expected from the function's return	Actual output What was actually returned from the function
Test_Insert	Insert one item, it should become the root.	Insert	passed	Root item address = 1	Root item address = 0x769960
Test_Insert	Insert one item, the tree size should be 1	Insert	failed	m_nodeCount = 1	m_nodeCount = 0

3. Unit Tests

```
StartTest( "Check count for tree size 5" ); {
    BinarySearchTree<char, string> bst;
    bst.Insert( 'B', "" );
    bst.Insert( 'A', "" );
    bst.Insert( 'C', "" );
    bst.Insert( 'D', "" );
    bst.Insert( 'E', "" );

    int expectedOut = 5;
    int actualOut = bst.GetCount();

    Set_ExpectedOutput( "count", expectedOut  );
    Set_ActualOutput  ( "count", actualOut );

    if ( actualOut != expectedOut )
    {
        TestFail();
    }
    else
    {
        TestPass();
    }
}  FinishTest();
```

Here's an example test:

Insert 5 items into a binary tree

Our expected size is 5

Get the actual size returned by the function

If the actual size doesn't equal the expected size, display an error.

Otherwise, the test passed.

3. Unit Tests

```
4   float PricePlusTax( float price, float tax )
5   {
6       return price + price * tax;
7   }
8
9   void Test_PricePlusTax()
10  {
11      // Test 1
12      float input_price = 10.00;
13      float input_tax = 0.09;
14      float expected_output = 0.9;
15      float actual_output = PricePlusTax( input_price, input_tax );
16
17      if ( actual_output == expected_output )
18      {
19          cout << "Pass" << endl;
20      }
21      else
22      {
23          cout << "Fail" << endl;
24      }
25  }
```

Your tests can be very simple; just a function that passes in some input and checks the output and displays an appropriate message.

If you design your programs with function “input/output” in mind, your functions will be more testable.

Conclusion

This is just a high-level overview of unit tests. We will practice unit tests more in a lab.

I also suggest that you try writing unit tests for future labs to help you validate your work and practice writing unit tests. :)