Test Driven Development

CS 16: Solving Problems with Computers I
Lecture #9 PRE-RECORDED

Ziad Matni
Dept. of Computer Science, UCSB

```
int main(int argc, char *argv[])
{
    if (argc > 1)
        filename = argv[1];
    ifstream setIn(filename);
    ifstream vecIn(filename);
    set<string> wordSet = getWordSet(setIn);
    vector<string> wordVec = getWordVec(vecIn);
    map<string, string> wordMap = generateMap(wordVec);

    string name = filename.substr(0, filename.size() - 4);
    string setFilename = name + "_set.txt";
    string vecFilename = name + "_vec.txt";
    string mapFilename = name + "_1_1.txt";

    // Writes set file
    ofstream setOut(setFilename);
    for (set<string>::iterator it = wordSet.begin(); it != wordSet.end(); it++)
    {
        setOut << *it << endl;
    }
    setOut.close();

    // Writes vector file
    ofstream vecOut(vecFilename);
    for (int i = 0; i < wordVec.size(); ++i)
    {
        vecOut << wordVec[i] << endl;
    }
    vecOut.close();

    // Writes to map
    ofstream mapOut(mapFilename);
    printMap(wordMap, mapOut);
    mapOut.close();

    // Generate and print random string
    string str = "";
    for (int i = 0; i < 100; i++)
    {
        cout << wordMap[str] << " ";
        str = wordMap[str];
    }
    cout << endl << endl << endl;

    // Generate more intelligent map
    map<string, vector<string>> wordVecMap;
    str = "";
    for (int i = 0; i < wordVec.size(); i++)
    {
        wordVecMap[str].push_back(wordVec[i]);
        str = wordVec[i];
    }

```

TDD

- TDD is a software development process
  - Very popular in industry

- Relies on the repetition of very short cycles
  - Come up with requirements (not code) first
  - Turn requirements into very specific test cases (still not code)
  - Now write code and test it with the test cases
  - Improve code until the test(s) pass

TDD “Lite”

- Write test code and actual code side by side - so your implementation is always tested
  - Professionals insist that you do the test code FIRST, then the actual code
- Write the simplest test case and make your code pass that case
- Write another test case, expect your code to fail, see it fail, then add code to pass that test case (and the previous one, naturally…)
- With every new test case, we have to make sure that all our previous tests still pass - this is a great way to make sure that things that were working before are not broken by new code!

Example

- We’ll write test cases that describe what the intended behavior of a unit of software should do BEFORE implementing the functionality
  - Define the requirement of this piece of software.

- Let’s say that I want to write a function that returns a string
  - See example on next slide…

Example of a Function Specification

Requirement / Spec:

- Write a function that “draws”, in ASCII characters, a square using the “*” character if you give it an integer input for the side size

Example:

- **`drawSquare(5)`** would return:

```
*****
*****
*****
*****
*****
```

First Step: Write a Test for this Requirement

- BEFORE you write the code! You will want 2 things:
  1. Something to check on expected value vs. actual value
  2. Something to run this check on a test of the function

```
void assertEqual(string expected, string actual, string message = "") {
    if (expected == actual) {
        cout   << "PASSED: " << message << endl;
    } else {
        cout   << "\tFAILED: " << message << endl;
        cout   << "Expected: [\n" << expected << "]\n"
               << "Actual: [\n" << actual << "]\n";
    }
}
```

Example: **assertEqual()**

- Example run: `assertEqual("**\n**\n", "*\n*\n", "testLength : 2")`

Would result in:

```
      FAILED: testLength : 2
Expected: [
**
**

]
Actual: [
*
*

]
```

Example: **testDrawSquare()**

- Now the 2nd part…
  - Something to run this check on a test of the function

NOTE!!!!
We haven’t defined `drawSquare()` yet!!!

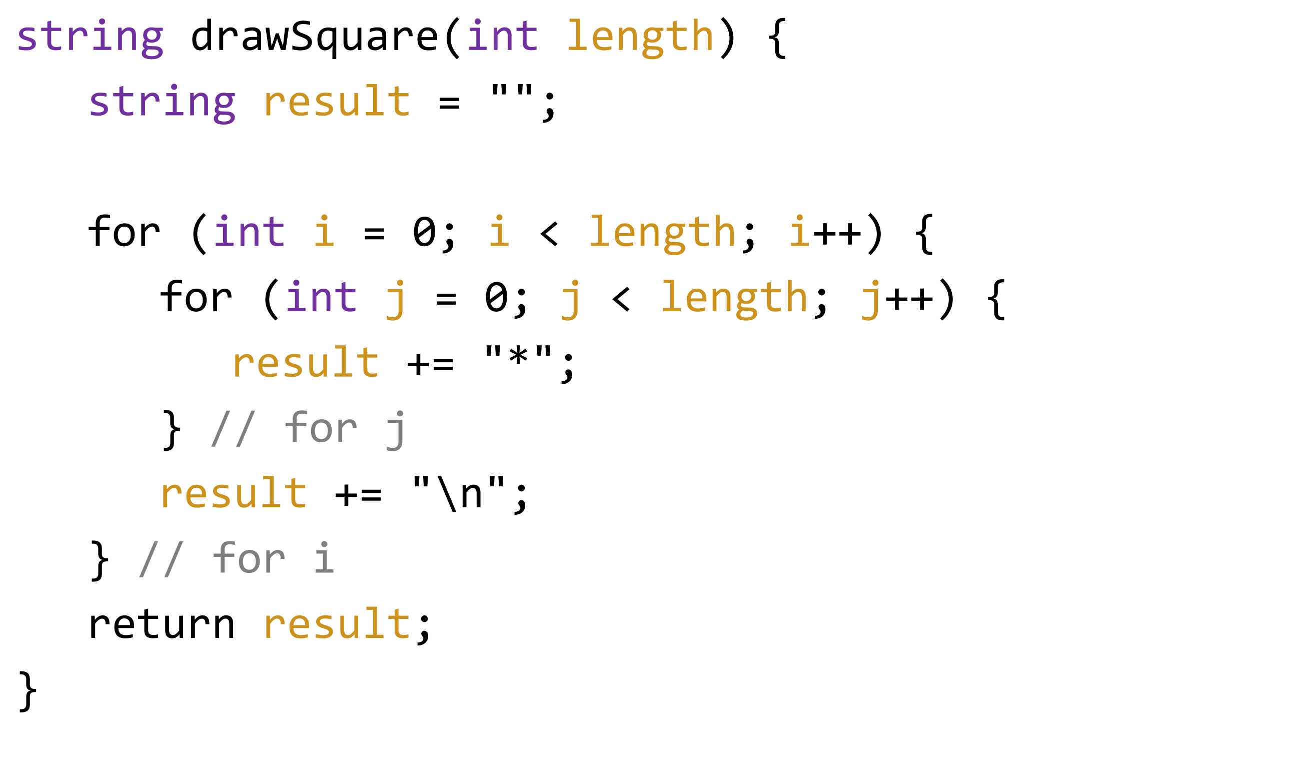
```
void testDrawSquare() {
    string expected1 = "**\n**\n";
    string actual1 = drawSquare(2);

    assertEqual(expected1, actual1, " testLength: 2");

    string expected2 = "***\n***\n***\n";
    string actual2 = drawSquare(3);

    assertEqual(expected2, actual2, " testLength: 3");
}
```

NOW! Write the Code for **drawSquare()**



```
string drawSquare(int length) {
   string result = "";

   for (int i = 0; i < length; i++) {
      for (int j = 0; j < length; j++) {
         result += "*";
      } // for j
      result += "\n";
   } // for i
   return result;
}
```

AND FINALLY!!! TEST IT!!! ☺

Setup:

Let’s assume good multi-file process, for example:

- Your **`drawSquares()`** declaration is in a file called “**`drawShapes.h`**”
- Your **`drawSquares()`** definition is in a file called “**`drawShapes.cpp`**”
- Your main program that runs this func. is in a file called “**`programXYZ.cpp`**”
  - Maybe it has other things in there that it runs too (we’re just focused on **drawSquare**)
  - The demo I will show you has an additional function called **drawRightTriangle()**
    - Leave that for you to explore… ☺

AND FINALLY!!! TEST IT!!! ☺

Setup:

- Let’s assume good multi-file process for your TEST SUITE TOO!!!

- Place your **`assertEqual()`** declaration in a file called “**`tdd.h`**”
- Place your **`assertEqual()`** definition in a file called “**`tdd.cpp`**”
- Place your **`testDrawSquare()`** definition in a file called “**`testDrawShapes.cpp`**”
  - Maybe it has other things in there that it tests too (we’re just focused on **drawSquare**)

What Will This Look Like?

- 6 files to deal with (3 for the program, 3 for the test suite)
- Create a **makefile** to help you compile and run them all

- We will now do a DEMO! Take Notes!

DEMO!!

- These files are all available on our website too

</LECTURE>