Outline

- Promoting the level of abstraction with which we are considering our problem solution.
- Test driven development
- Friday is work-on-homework day.

Which is faster?

- 12+6

+

Let's look at some starter code for HW1. I

- Use this file every time, especially in homework.

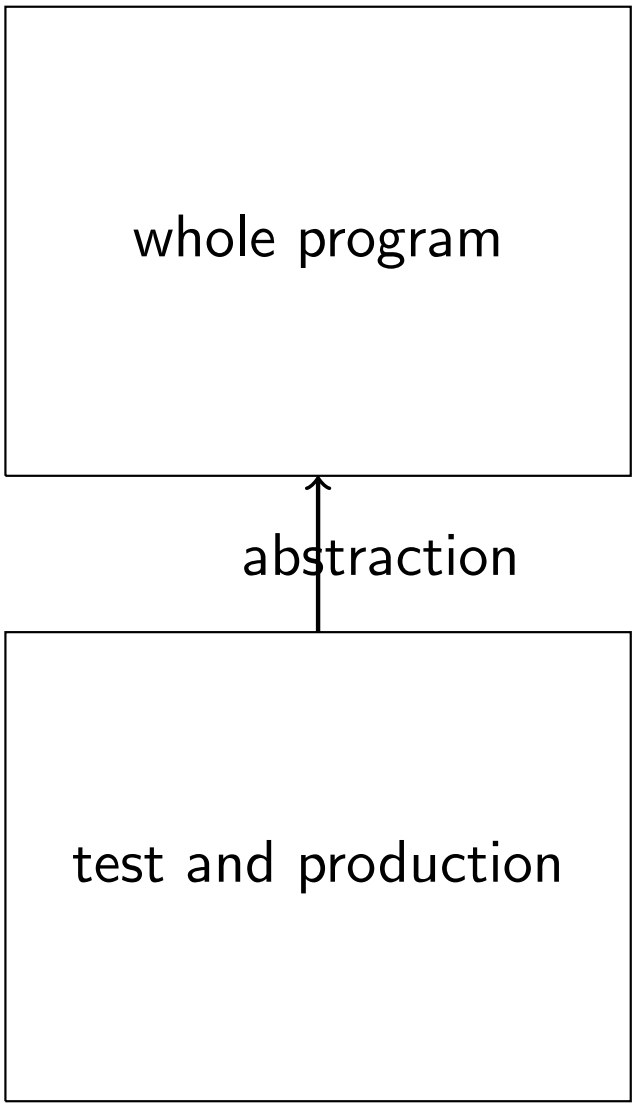
```
#include <stdio.h>  //This is a system file, we are using
#include <stdlib.h>
#include "tests.h"   //This is a developer-created file. Us
#include "production.h" //ditto
#include "TMSName.h"  //Use your own name.

int main(int argc, char* argv[]) {
puts("!!!Let's do HW1!!!");
if(tests())
{
production(argc, argv);
}
else
{
```

Let's look at some starter code for HW1. II

```
puts("Tests did not pass.");
}
return EXIT_SUCCESS;
}
```

Code structure can be depicted.



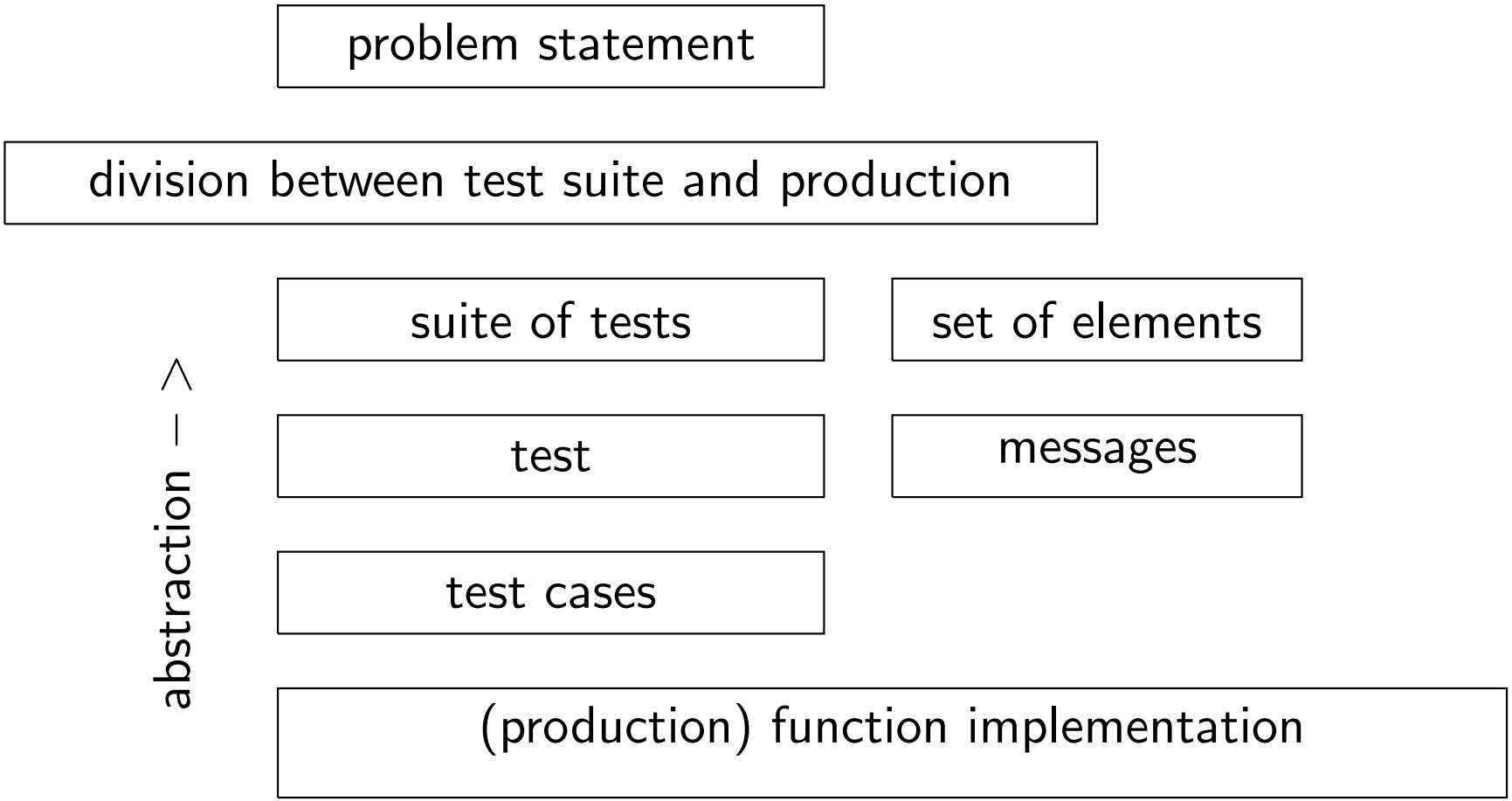
The file has two levels of abstraction.

- The upper level of abstraction is: This is the solution to the problem.
- The lower level of abstraction is: We divide the solution into the test suite, and the production code.
- It can make the code easier to understand if there are not too many different levels of abstraction in one place.

At the level of abstraction that is above which specific tests and which specific production code, we have the ideas of the test suite, and running it before attempting the production code.

- People think faster, and often more accurately, at higher levels of abstraction.
- We want to use this when we are developing code.
- We will use several levels of abstraction,
  such as statements, functions, groups of functions.
- We will move among them from high to low and back again.
- We will strive to remain at a higher level of abstraction for the speed.

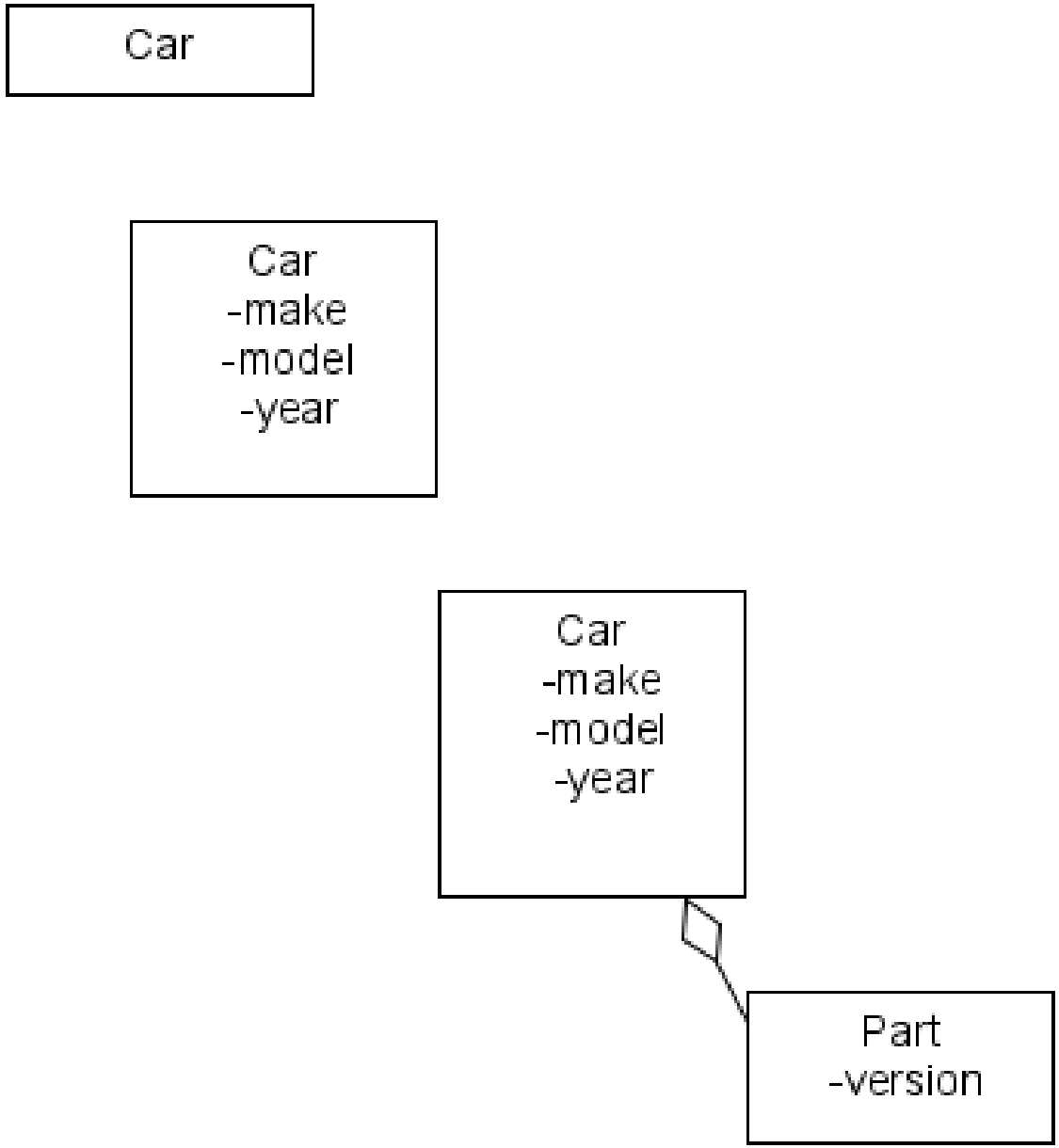
Code structure can be depicted.



We have everyday examples of levels of abstraction.

- Call the automotive service station, to make an appointment for repairing the car.
- Here the level of abstraction is “car”.
- The kind of work to be done helps determine the length of the appointment,
  and a day and time can be set.
- If the service involves replacement of certain parts, the make, model and year of the car become relevant.
- If the particular item being replaced has versions, within the model and year, that becomes relevant.
- Once the parts are ordered, installed and checked, we return to a higher level of abstraction.
- “Your car is ready.”

Car levels of abstraction can be depicted.



We traverse levels of abstraction when we develop (including test) code.

- We analyze the problem statement.
- We design a sequence of interactions among elements, that make up a solution.
- Moving to a lower level of abstraction,
- we develop each individual element's role in one interaction (including tests).
- Test cases operate at a relatively low level of abstraction.
- Function implementation is at a yet lower level of abstraction.
- Once a function is satisfactorily tested, we think of it as a unit, that is, at a higher level of abstraction.
- At the level of abstraction where functions are units, we combine the functions (This can imply more tests.)
- When all of the functions have been combined, the solution is a unit.

Let's view an example of a problem statement.

- There is a spreadsheet file, with multiple tabs.
- Each of the several tabs holds a sheet of the spreadsheet file, and each sheet has rows and columns of data.
- This is an array exercise, so the datatype of the elements filling the spreadsheet cells is the same, and we're using double.
- The number of rows on each sheet is the same.
- The number of columns on each sheet is the same.
- Find the mean and variance for each sheet, and
- for the file as a whole.

Here's the figure:

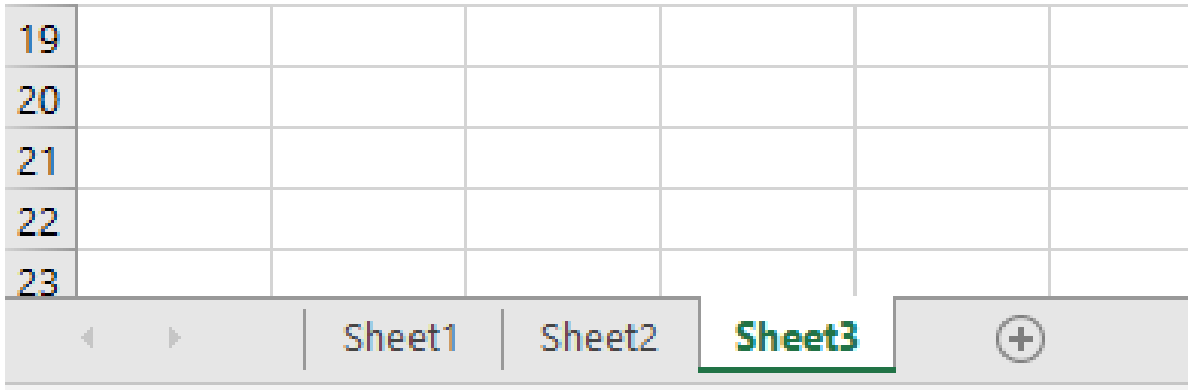
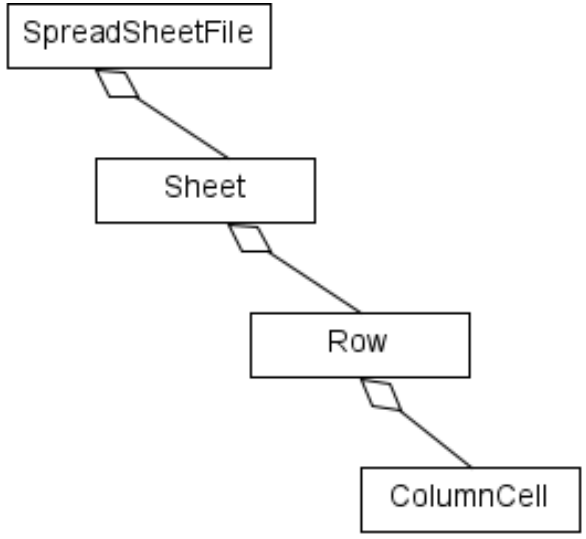
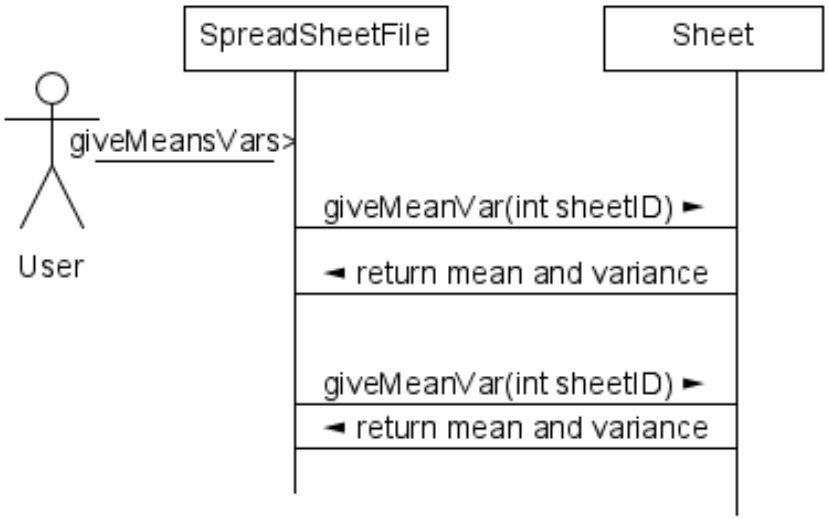
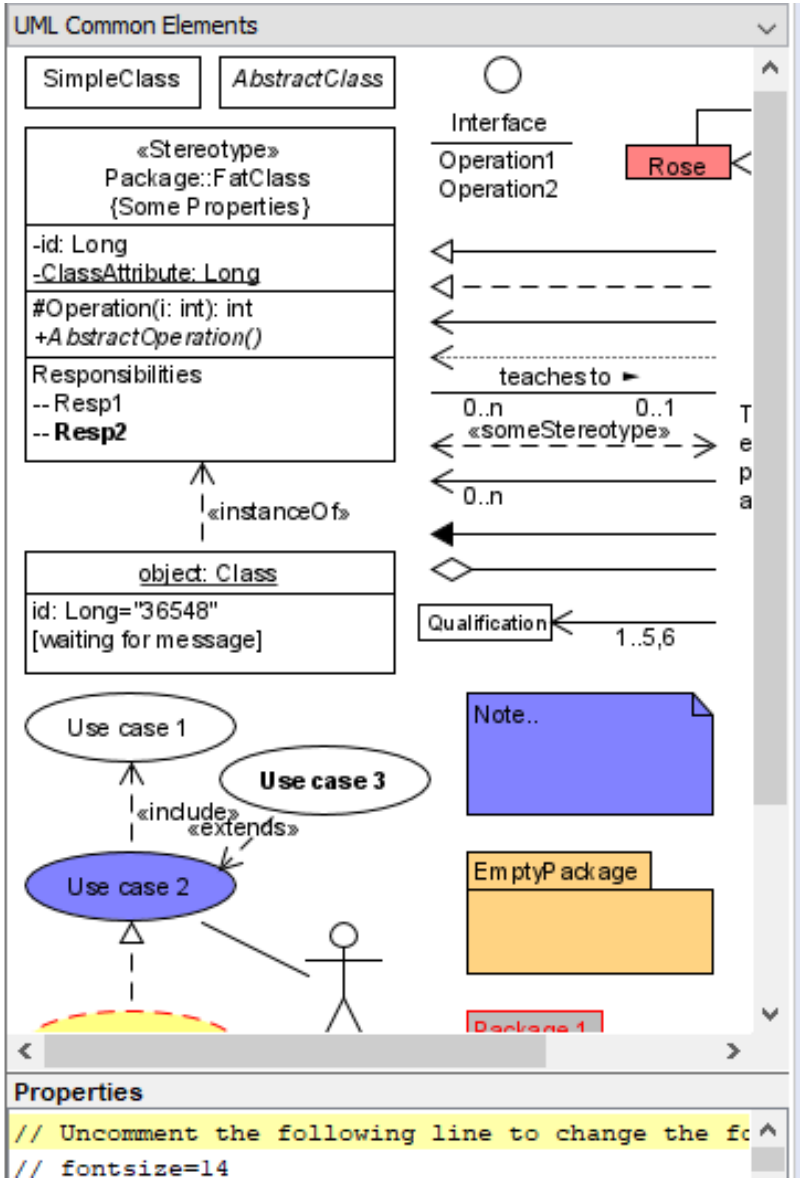


Figure: There are multiple sheets, each sharing a common number of rows, and a common number of columns.

We can approach the design with a sequence diagram.







Levels of abstraction help analyze this problem.

- The highest level of abstraction found in this problem is a file as a unit.
- The next lower level of abstraction is the sheet; we say a file is composed of some number of sheets.
- The next lower level of abstraction is either row or column, we choose row.
- (This is because columns are often items of data, and rows are often measurements, patients, etc.;
- instances that have values for each column).
- This leaves column cell as the lowest level of abstraction.
- We could hold a structure at each cell, which would add depth to the levels of abstraction.
- We will stop arbitrarily at the cell. In this problem, the cell contains a double, i.e., a rational number.

The sequence diagram shows the messages.

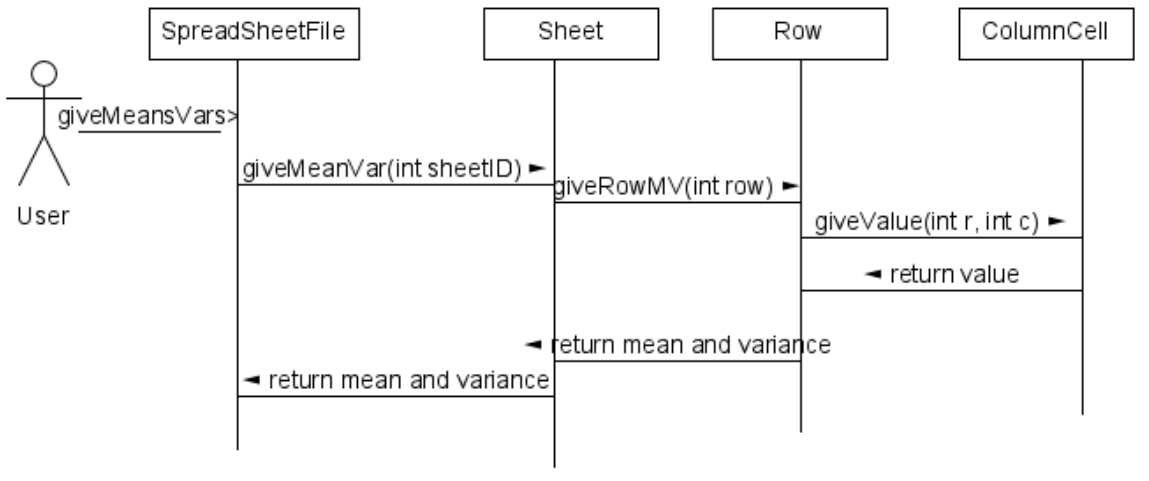
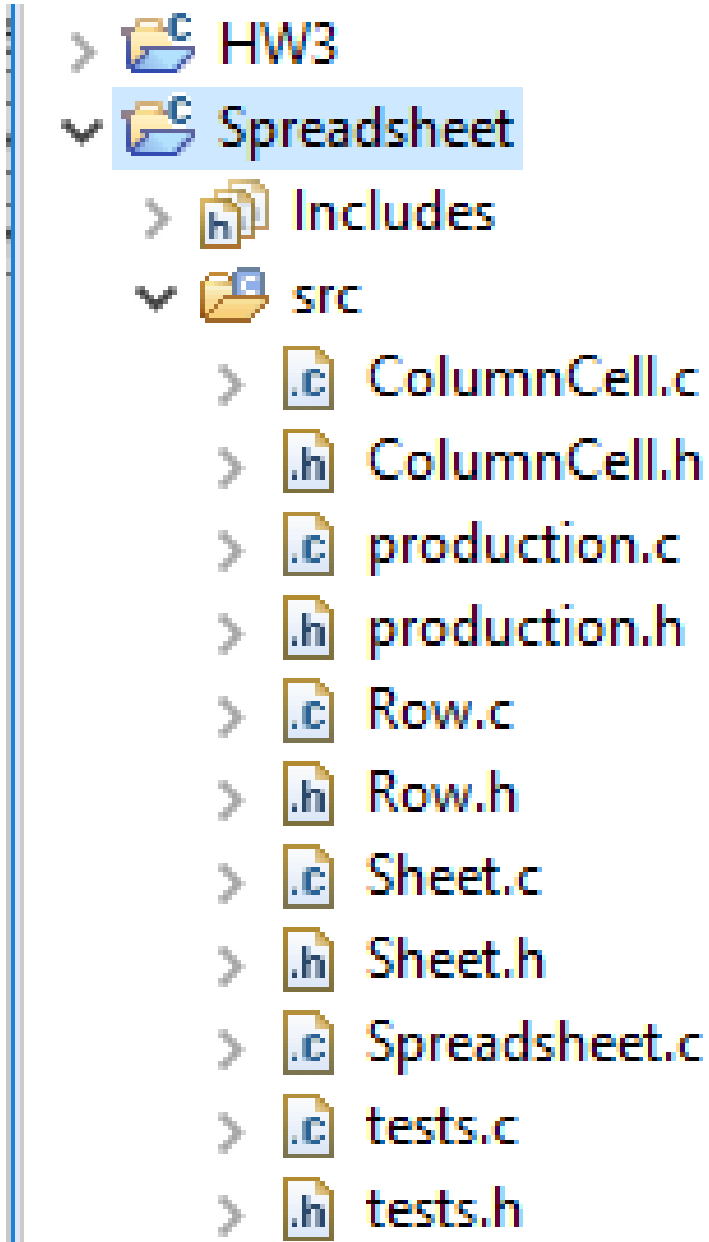


Figure: Each message, with its return, implies a function call.

The structure of the solution is reflected in the structure of the code.



The nouns of the problem, being identified, appear in the top row of the sequence diagram. I

- We intend to organize the design effort using the principle that objects exchange messages.
- The "objects" come from the nouns mentioned earlier.
- The next thing to design is a sequence of messages.
- This is occurring at a level of abstraction higher than code.
- In this problem, we can imagine a spreadsheet file addressing individual sheets,
- requesting of each sheet, what its mean and variance are.
- Peeking into a lower level of abstraction, we anticipate that a sheet will need a method to determine these properties.
- A message implies a function invocation, and therefore a function,
- A function implies at least one test function, and

The nouns of the problem, being identified, appear in the top row of the sequence diagram. II

- a test function implies at least two test cases.
- These functions and test functions exist at a level of abstraction below the messages.
- The test cases exist at a level of abstraction below the test.
- The code of setting up, applying and comparing the answer with the known correct answer is at a lower level of abstraction than the test case.
- The code of the function being tested is at a level of abstraction below the test/test case code.
- There are potentially many levels of abstraction, and the ability to navigate nimbly through these is learned by practice. Awareness helps. Thinking directly in code might be fun, but it does not scale well.

Using the messages to infer functions which imply tests.

- There is a message arriving at Sheet, asking mean and variance.
- So we will have a function in Sheet, that returns a mean and variance.

```
typedef struct
{
    double mean;
    double variance; //cannot be negative, so negative means bad arguments
} MeanAndVariance;
```

Let's apply our abstraction idea to datatypes.

- C has primitive datatypes: int, float, double, sort of bool, char, etc.
- (sort of means we use an include file to get it)
- We can make user defined datatypes, with typedef struct, as you saw above.
- The meanAndVariance is a datatype, a unit, at a higher level of abstraction.
- The meanAndVariance datatype, at a lower level of abstraction, has components.

We will use the messages to infer functions.

- There is a message arriving at Sheet, asking mean and variance.
- So we will have a function in Sheet, that returns a mean and variance.
- That function will have a function prototype.
- We will put function prototypes in .h files.
- MeanAndVariance giveMeanAndVariance(double* ourData, int nSheets, int nRows, int nCols, int sheetNum);

We will use the function (prototype) to infer a test, and some test cases. I

```
#include "tests.h"
#include "production.h"
#include "Sheet.h"

bool tests()
{
bool answer = false;
bool ok1 =  testSheetMandV();
answer = ok1;
return answer;
}

bool testSheetMandV()
{
```

We will use the function (prototype) to infer a test, and some test cases. II

```
bool ok = false; //this is for the logical AND of the test
printf("running testSheetMandV\n");
//There should be an array of data, that corresponds to the
//Each sheet should be able to access at least its own data
//Let's say that the sheet functions are told which sheet,
//Let's use 0 to some positive number as the appropriate ra
//Therefore we can make a test case with a negative sheet n
//and it will be the job of the function to reject such a n

        int nSheets = 2;
        int nRows =3;
        int nCols= 4;
        double ourData[nSheets][nRows][nCols];
        for(int sheet = 0; sheet<nSheets; sheet++)
        {
```

We will use the function (prototype) to infer a test, and some test cases. III

```
        for(int row = 0; row<nRows; row++)
        {
            for(int col = 0; col<nCols; col++)
            {
                ourData[sheet][row][col]= 5; //mean wi
            }
        }
    }
    double* dp = (double*)ourData;

    bool ok1 = false;
    int sheetNum = -3;
    MeanAndVariance mv = giveMeanAndVariance(dp, nSheet
    if (mv.variance <0)
    {
```

We will use the function (prototype) to infer a test, and some test cases. IV

```
            printf("Negative sheet number test passed.\n");
            ok1 = true;
        }
    bool ok2 = false;
    sheetNum = 0;
    mv = giveMeanAndVariance(dp, nSheets, nRows, nCols,
    if ((mv.variance == 0) && (mv.mean == 5))
    {
            printf("Positive sheet number test passed.\n");
            ok2 = true;
    }

    ok = ok1 && ok2;
    return ok;
}
```

Was that second test case very useful?

- Zero is a valid sheet number.
- It is convenient for us to make that so, because we'll use an array
- as our internal representation of the data from the file.
- and arrays in C are indexed starting with zero.
- Is the correct answer for variance 0 for this scenario?
- Yes, because every value is the same, there is 0 variance.
- But, a function that always returns 0 would pass that test.
- We should also have a test case whose correct answer is not zero.
- Care must be taken with test cases, or else
- the test cases will not be giving us the information we need.

What makes a test case valuable?

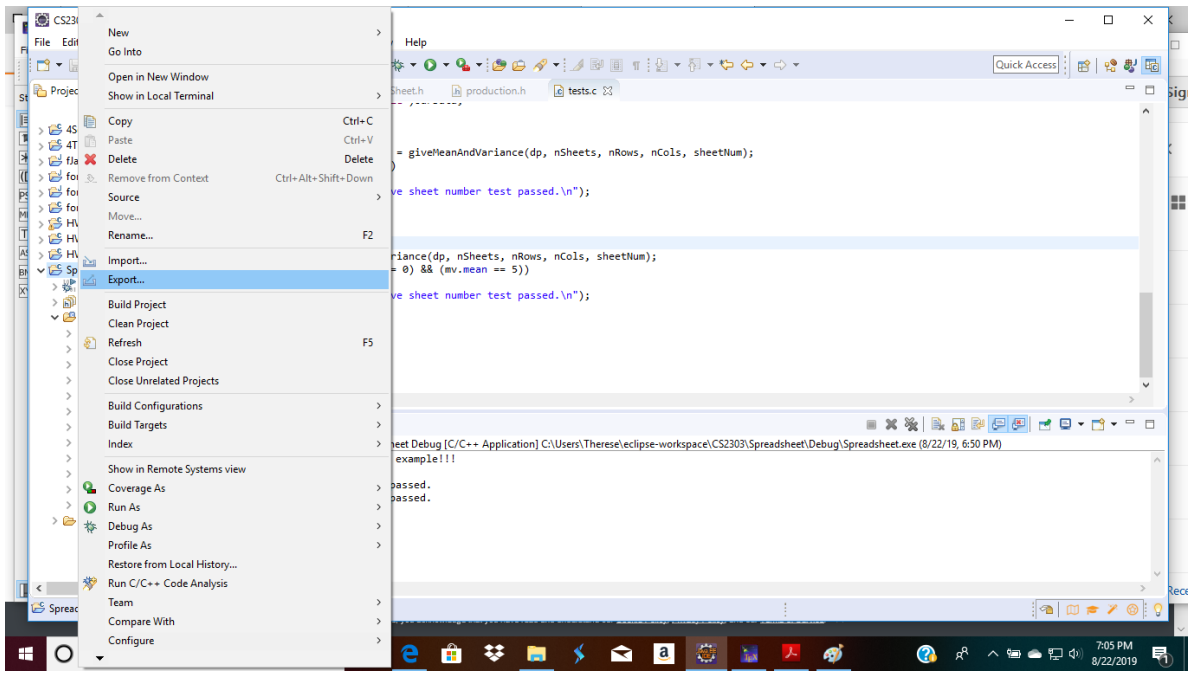
- The idea is that a test case that should pass does pass,
- and a test case that should fail does fail,
- and by a suitable collection of test cases,
- we obtain confidence that our implementation is correct.
- See the bowling game kata, if you have not already.
- What makes a test case valuable is that it informs us about the correctness of our implementation.

What is a good test case in this example?

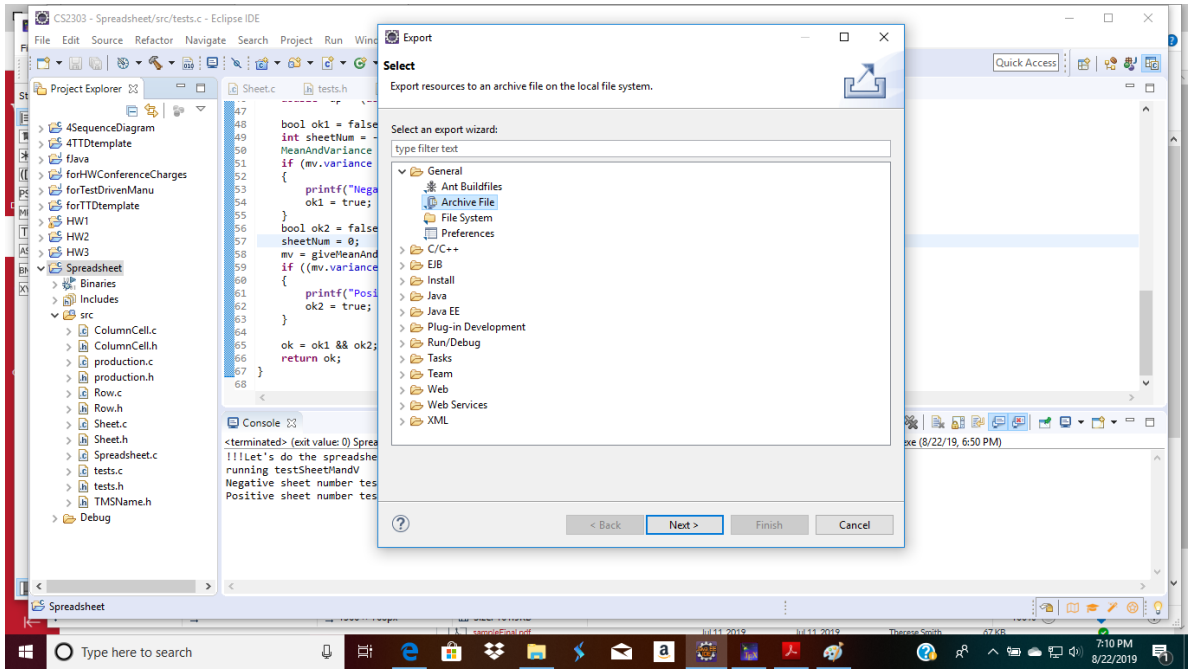
- We need to get the right mean, and the right variance,
- for a part of the array, that had interesting values.

Homework 1 is not due until September.

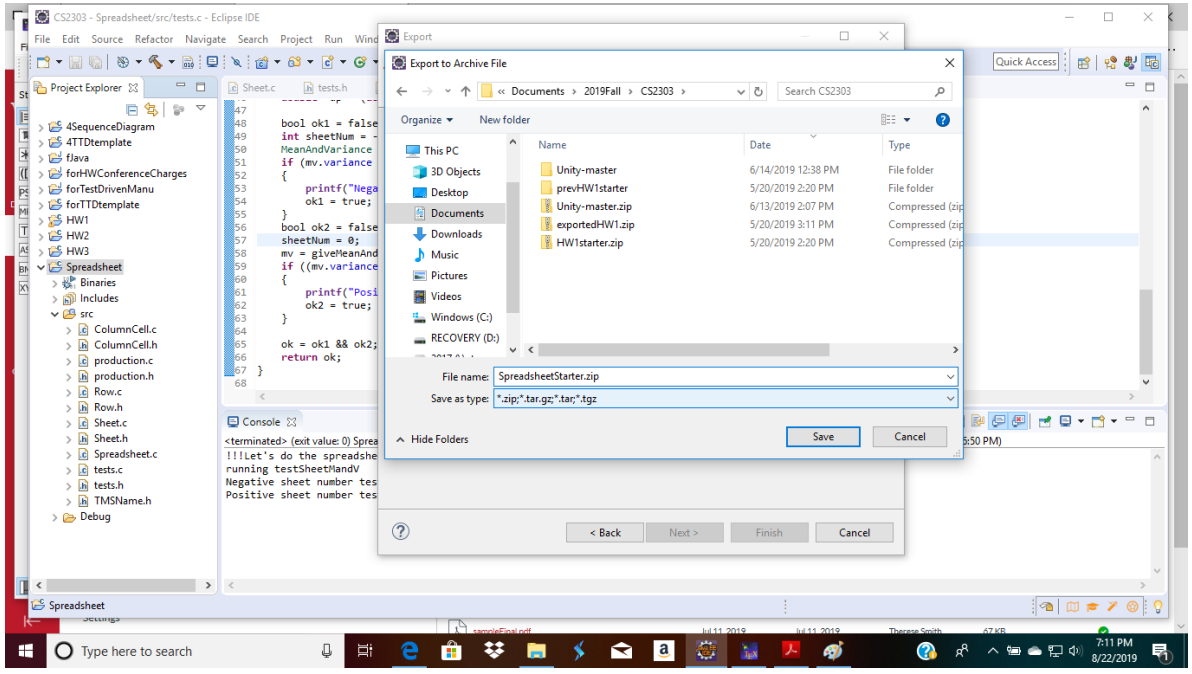
- We will talk about starter code.
- Most weeks, you will want to obtain the starter code for that week's lab and
- the subsequent homework.
- I shall export code. (I used right click on the project name.)



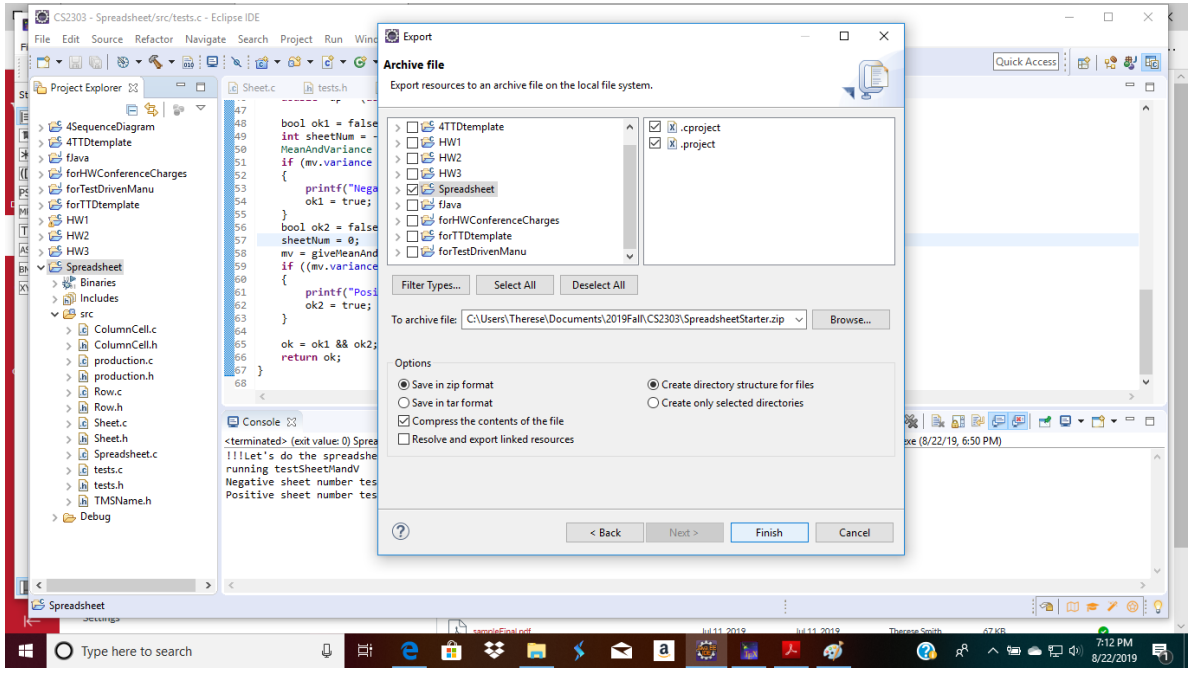
Export to the file system. I



Export to the file system. II



Export to the file system. III



Upload to Canvas.

- When you submit a homework file,
- you will perform a similar export and upload.