David needs to test a function that compares dates (in C) with the following specification:

/\* returns 0 if date1 = date2, -1 if date1 before date2, 1 if date1 after date2,  
-2 if any element of any date is invalid;  
valid years are all integers (<0 = BC); valid month is 1-12; valid day is  
1-{28,29,30,31} depending on the year and month \*/  
int compdate(int year1, int month1, int day1, int year2, int month2, int day2);

He proposes a random testing scheme, running 2,000,000 tests where each test is  
performed by the following function:

int oneTest() {  
int y1 = random\_int(); /\* returns a random 32 bit value \*/  
int y2 = random\_int();  
int m1 = (random\_int() % 11) + 1;  
int m2 = (random\_int() % 11) + 1;  
int d1 = (random\_int() % 30) + 1;  
int d2 = (random\_int() % 30) + 1;  
int c1 = compdate(y1,m1,d1,y2,m2,d2);

int c2 = compdateOracle(y1,m1,d1,y2,m2,d2);

Assert(c1 == c2, "Issue with compdate function");

}

Assuming that compdateOracle is a perfect function without any bug to compare dates, Find out 3 issues with this random testing scheme. For each of the issue found, propose an improvement. Mention issue in one line and improvement proposal in less then 3 lines. Find out logical issue, don't look for typos and coding errors.

Your Answer:

1.

Issue:

The days validity is not dependent on the month, as it should be. This test does not check if a particular month has the correct amount of days.

Improvement:

Create a switch statement that process the month numbers and calculates the correct amount of days for that month.  For example: If the month number passed into the switch statement is a month with 31 days, then it goes to a section of the switch statement that calculates days up to 31 and if the month passed in is a month with 29 days, then it goes to a section of the switch statement that calculates days up to 29.

2.  
Issue:  
The Assert function: Assert(c1 == c2, "Issue with compdate function"). It does not check for all possible outcomes.

Improvement:  
I would remove the assert function all together and replace it with what the oneTest() descriptions calls for, which is comparing c1 to c2 to see if it's equal to, less than, or greater than. This comparison could be done with 3 if statements. For example if c1 is greater than c2 then the result is a 1 or if c1 is less than c2 the result is a -1; finally if they are equal than the result is 0.

3.

Issue:

There is NO return statement.The directions state that the function will be called 2 million times and each call must have something returned.

Improvement:

At the end of the function, return the compared result between c1 and c2. For example: If (c1 > c2) { return 1;} or if(c1 < c2) {return -1;} or if(c1==c2){return 0;}

Consider the following design of game of chess.

**Chessobject chessboard[][]**

**chessboard** is a two dimensional array of size 8 X 8. It represents a chessboard in the game of chess. It holds chess objects.

**chessboard[2][3] = whitepawn1** means currently on my chessboard, I have whitepawn1 on cell 2,3.

**chessboard[6][3] = blackpawn4** means blackpawn4 on cell 6,3.

**chessboard[5][1] = null** means cell 5,1 is empty.

By defining all 64 cells on the chessboard, we have a state of game of chess.

As we will play the game of chess, we will remove objects from the game of chess.

**Chessobject discardpile[]** is a one dimensional array of 32 size that holds removed chess object from the game of chess.

Combinely chessboard[][] and discardpile[] defines a valid state of game of chess. Though the definition is not complete, but let’s assume that it is a valid state of game of chess for rest of our discussion.

I define **RookMove** method as follows. Remember, rook moves in horizontal and vertical direction and cannot jump over a piece.

bool RookMove(ChessObject[][] chessboard, ChessObject[] discardpile, ChessObject rookObject, int a, int b, int x, int y)

first two parameters are current state of game. Assume that as third parameter, I will always pass correct rook object. whiterook1, whiterook2, blackrook1 or blackrook2.

a,b is the source position of the rook that was passed to the method. Assume that passed rook is indeed at position a,b.

x,y is the destination position of the rook. The cell where the rook is supposed to move.

**If the move is legal, necessary changes will be made to chessboard and discardpile and method will return true.**

**If the move is illegal, no changes will be made to chessboard and discardpile and method will return false.**

It is important to note that the changes in **chessboard** and **discardpile** will help us to determine if RookMove is doing its job correctly or not.

Consider one particular move.

**RookMove(chessboard, discardpile, whiterook1, 2,7,2,3)**

I am trying to move rook-whiterook1 from 2,7 position to 2,3 position.

**I want you to test this particular call. So, don't spend time doing a generic testing of RookMove. Just tell me how you will test this particular call.**

**RookMove(chessboard, discardpile, whiterook1, 2,7,2,3)**

As you can see, I am trying to move rook in **horizontal direction**.

**Provide 3 test cases (unit test - pseudo code) for testing the particular call. Describe your test using verbal scenario first and then write pseudo code. Pseudo code should look like “code”. Assert appropriately.**

Your Answer:

1.  
To see if RookMove() function worked I can check to see if the Whiterook1 has moved from  
it's original spot. I can also check if it reached it's destination.  
  
code:  
RookMove(chessboard, discardpile, whiterook1, 2,7,2,3)  
  
//if not equal then the rook moved  
if(chessboard[2][7] != whiterook1)     
    print "Whiterook1 moved, checking if reach it's destination of 2,3"      
      
    // Checking to see if it reached it's destination  
    if(chessboard[2][3] == whiterook1)          
        print "\*\* PASSED \*\* Whiterook1 arrived at destination --> 2,3"          
    else  
        print "\*\* FAILED \*\* Whiterook1 did not arrive at destination --> 2,3"  
else  // if true the rook did not move  
    print "\*\* FAILED \*\* Whiterook1 did not move"

      
      
  
2.   
To see if RookMove() worked I can also check if the discardpile[] has increased in size.   
When the function calls and the rook moves, if any enemy piece is in the path, it will   
be sent to the discard pile. Checking the final destination will not always be accurate  
if whiterook1 moved, because if it encounters an enemy it will stop at that enemies   
chossboard[][] location.

code:  
int discardBeforeSize = sizeof(discardpile)  
RookMove(chessboard, discardpile, whiterook1, 2,7,2,3)  
int discardAfterSize = sizeof(discardpile)  
  
if (discardAfterSize > discardBeforeSize)  
    print "\*\* PASSED \*\* whiterook1 moved and destroyed an enemy piece."  
else if (discardAfterSize == discardBeforeSize)  
    //Checking to see if whiterook1 reached it's destination w/o encountering an enemy.   
    if(chessboard[2][3] == whiterook1)  
            print "\*\* PASSED \*\* Whiterook1 did not encounter an enemy piece and arrived at destination --> 2,3"  
    else  
            print "\*\* FAILED \*\* Whiterook1 did not arrive at destination --> 2,3"

3.   
To see if RookMove() worked, I can also check if whiterook1's current location is NULL   
or not. If it is null, then I know that the whiterook1 moved and if it's not null  
then I know it didn't work.   
  
code:  
RookMove(chessboard, discardpile, whiterook1, 2,7,2,3)  
if chessboard[2][7] == null  
    print "\*\* PASSED \*\* The whiterook1 moved from the location 2,7  
else  
    print "\*\* FAILED \*\* The whiterook1 did not move"

Design a manual testing scheme for compose functionality of gmail. Consider both good cases and bad cases. Be thorough in verifying the output. Think of all the possible ways an email can be composed and everything that can possibly go wrong and how will you verify it manually.

Your Answer:

// Used to test if the gmail compose function properly loaded the other functions that make it  
// it work correctly. testing for spellcheck, word processing, and send button capabilities.   
int testProperComponentsOfComposeFunction()  
{  
    int engagedCorrectly, sendCorrectly, spellCheckCorrectly, wordProcessingCorrectly;  
    string testMsg = "This is to test the send function";  
    string spellingError = "Helo";  // should be spelled "Hello"  
    gmailCompose gCompose = new gmailCompose();  
      
    // If 1 returned then compose function engaged properly  
    // If -1 returned then compose function did not engage properly   
    engagedCorrectly = gCompose.isValid();  
    if(engagedCorrectly == 1)  
        print "\*\* PASSED \*\* Gmail entered the compose state "  
    elif (engagedCorrectly == -1)  
        print "\*\* FAILED \*\* Gmail did not eneter the compose state  
    else  
        print ("\*\* FAILED \*\* Gmail returned an unknown value of %d", engagedCorrectly)  
        return engagedCorrectly; // Gmail didn't load correctly, so the other tests will not pass  
          
    spellCheckCorrectly = gCompose.spellCheck(spellingError);  
    // If 1 returned then spelling function engaged properly  
    // If -1 returned then spelling function did not engage properly   
    if(spellCheckCorrectly == 1)  
        print "\*\* PASSED \*\* Compose entered the spelling state "  
    elif (spellCheckCorrectly == -1)  
        print "\*\* FAILED \*\* Compose did not enter the spelling state"  
    else  
        print ("\*\* FAILED \*\* spelling returned an unknown value of %d", spellCheckCorrectly)  
          
          
    // If 1 returned then word processing function engaged properly  
    // If -1 returned then word processing  function did not engage properly       
    wordProcessingCorrectly = gCompose.wordProcessing(testMsg);  
    if(wordProcessingCorrectly == 1)  
        print "\*\* PASSED \*\* Compose entered the word processing state "  
    elif (wordProcessingCorrectly == -1)  
        print "\*\* FAILED \*\* Compose did not enter the word processing state"  
    else  
        print ("\*\* FAILED \*\* word processing returned an unknown value of %d", wordProcessingCorrectly)  
  
      
    sendCorrectly = gCompose.send(testMsg);  
    // If 1 returned then send function engaged properly  
    // If -1 returned then send function did not engage properly   
    if(sendCorrectly == 1)  
        print "\*\* PASSED \*\* Compose entered the send state "  
    elif (sendCorrectly == -1)  
        print "\*\* FAILED \*\* Compose did not enter the send state"  
    else  
        print ("\*\* FAILED \*\* Send returned an unknown value of %d", sendCorrectly)              
      
    return engagedCorrectly, sendCorrectly, spellCheckCorrectly, wordProcessingCorrectly;  
              
}  
  
  
  
//Used to test if the compose function loaded within the company mandated time of 30ms  
void testComposeStallOnLoading()  
{  
    //Standard load time is 30ms, with a timeout of 60ms  
    int exceptedTime = 30ms;  
    int expire = 60ms;      
    int timer = Timer();    // Create timer object  
    int engagedCorrectly;      
      
    timer.begin();  
    gmailCompose gCompose = new gmailCompose(); // This engages the compose button  
    engagedCorrectly = gCompose.isValid();  
    timer.stop()  
      
    // If 1 returned then compose function engaged properly  
    // If -1 returned then compose function did not engage properly  
    if(engagedCorrectly == 1 && timer < exceptedTime)  
        print ("\*\* PASSED \*\* Gmail entered the compose state under expected time of %d", timer);  
    elif (engagedCorrectly == 1 && timer < expire)  
        print ("\*\* PASSED \*\* Gmail entered the compose state above expected time of %d", timer);  
    elif (engagedCorrectly == 1 && timer == expectedTime)  
        print ("\*\* PASSED \*\* Gmail entered the compose state at expected time of %d", timer);  
    elif (engagedCorrectly == 1 && timer >= expire)  
        print ("\*\* FAILED \*\* Gmail entered the compose state, but over the expire time of %d by %d", expire, (timer-expire));  
    elif (engagedCorrectly == -1 && timer < exceptedTime)  
        print ("\*\* FAILED \*\* Gmail did not enter the compose state, but the error occurred under the expected time of %d by %d", expectedTime, (expectedTime - timer));  
    elif (engagedCorrectly == -1 && timer < expire)  
        print ("\*\* FAILED \*\* Gmail did not entered the compose state, but the error occurred within the expire list of %d by %d", expire, (expire-timer));  
    elif (engagedCorrectly == -1 && timer == expectedTime)  
        print ("\*\* FAILED \*\* Gmail did not entered the compose state, but the error occurred at the expected time of %d", timer);  
    elif (engagedCorrectly == -1 && timer >= expire)  
        print ("\*\* FAILED \*\* Gmail did not entered the compose state and the expire was over time of %d was over by %d", expire, (timer-expire));  
    else  
        print ("\*\* FAILED \*\* Gmail returned an unknown value of %d, engagedCorrectly")     
}  
  
  
  
// Used to test if the Gmail compose function loaded or not. And to check any errors   
// that may be be returned.   
void testComposeFunctionEngaged()  
{  
    // If 1 returned then compose function engaged properly  
    // If -1 returned then compose function did not engage properly   
      
    gmailCompose gCompose = new gmailCompose(); // This engages the compose button  
    engagedCorrectly = gCompose.isValid();  
      
    if(engagedCorrectly == 1)  
        print "\*\* PASSED \*\* Gmail entered the compose state "  
    elif (engagedCorrectly == -1)  
        print "\*\* FAILED \*\* Gmail did not eneter the compose state"  
    else  
        print ("\*\* FAILED \*\* Gmail returned an unknown value of %d", engagedCorrectly)  
}