Logan Kling
CS362 Aburas
April 17, 2017
Assignment #1

Assignment One Report

Description:

This program checks a number out to 1000 decimals to see if it's rational by checking that no digit shows up sequentially more than 19 times & by checking that no sequence of 20 digits matches the next sequence of 20 digits. Yes, this is a bad algorithm for checking if a number is rational, but as the power of this algorithm is increased, its cost increases exponentially.

Excluding main(), this program has five functions: checkRational(), readIn(), printOutput(), takeFraction(), & takeRandom().

checkRational(): loops through a 1000 byte array containing a number from 0-9 in each index & looks for the varied digit repetition of a fraction such as 2/7. It does this by taking a user defined window (20 in this case) & making two of those windows & checking them at every overlap except complete overlap. If these windows mismatch at every overlap, the second window moves forward one space, while the first window moves forward a window in size. If any of these windows match, the loop is broken and the value of the index at which the windows overlap is returned. If no matching windows are found, this is repeated until the second window reaches the end of the array & -1 is returned denoting that the number is irrational.

readIn(): takes a numerator, denominator, and the 1000 byte array & checks to see if there's a sequence of the same digit repeating at least 20 times while it puts the result of the long division of the numerator & denominator into the array. If it finds no sequence of at least 20 sequential instances of the same digit, it passes the array to checkRational() to determine the rationality of the number.

printOutput(): returns nothing & is just a way of simplifying takeFraction(), & takeRandom() which call it because a large amount of their output & functionality is the same. This function will actually call checkRational() instead of takeFraction() & takeRandom().

takeFraction(): creates a 1000 byte array, calls readIn() & prints the denominator / the numerator before calling printOutput() to print the rest.

takeRandom(): creates a 1000 byte array and fills it with random digits from 0 to 9 while checking that no sequence of those numbers is a digit that's shown up sequentially more than 20 times, prints the number it's generated, & proceeds to call printOutput(). Any number generated by this function should be irrational, but calling this function enough times will produce some false positives.

main(): simply calls takeFraction() & takeRandom() a few times to test this program.

Bug #1: Line 28, column 57 is missing `- windowSize`. This small piece of code would prevent an array overflow with the arrays of randomly generated numbers. This won't cause an issue for rational numbers because the algorithm checking them will never reach the end of the array anyways.

Bug #2: Line 58, column 31 is missing an `=`, after the `>`. This doesn't really effect the end result, but fractions with a finite number of decimals will be rounded somewhat incorrectly. For example: ½ will round to 0.49999999999999999999999999999999999…, instead of 0.5.

Bug #3: Line 122, column 53 should have a bigger number, preferably one that scales, like the size of the array divided by eight. Because there only have to be 20 repetitions of one digit, takeRandom() will say it has produced a rational number approximately 1% of the time due to the size of the array. This is a very small issue, but takeRandom() should never produce a rational number.

Bug #4: Line 116 is downright missing an if statement which resets the variable 'temp' to -1 if it's less than 20. This is a problem because there only have to be two of the same digits at the end of the array for this to cause the function to say it has produced a rational number. This bug additionally causes takeRandom() to say it has produced a rational number approximately 8% of the time.

Utility:

This program is designed to determine the rationality of a given number. This program is designed for numbers with infinite decimals (because all numbers with a finite amount of decimals are rational) but will work for anything. It looks for repeating patterns in the decimals of that number out to the 1000th place, & will report that the number is rational if it finds any, & irrational if it doesn't.

Unfortunately, the algorithm behind this program is flawed, & of course it's technically 100% impossible to determine that an infinite decimal number is rational, but I think this does a pretty good job of it for being designed in a language as horrible as Java. I was surprised how few bugs I ran into while implementing this. Coming up with the algorithm was honestly the hardest part. I think I had less than 20 bugs in this project total, & the four that I implemented were the only tricky ones.

If I were to improve this the first thing I would do is code this in something like C, as efficiency really matters for something like this. With C I could use bitfields for every array byte, holding two numbers for each byte—something of which I might be able to use Cobalt for. This still wouldn't be the most efficient use of memory, but the accuracy for something like this is extremely important. The most practical use I can think for this would involve calculating whether numbers were rational out to a million digits & beyond.

The Test Cases:

Fourteen test cases were generated. The only errors caught where the exceptions resulting from ArrayIndexOutOfBounds Exceptions which happened on tests 1, 5, 8, & 14.

This tool may be able to find most bugs, but the bugs this tool finds are far more explicit & then the user must still determine how to fix such bugs themselves. I believe this tool would be very useful for increasing the testing speed of large programs, but it certainly shouldn't be the be all end all for testing a program. There will need to be some kind of white box testing as well as black box testing by beta users.