Software Testing Assignment 6

Cindy Berghuizen, Omar Pakker, Chiel Peter, Maria Gouseti

October 8, 2013

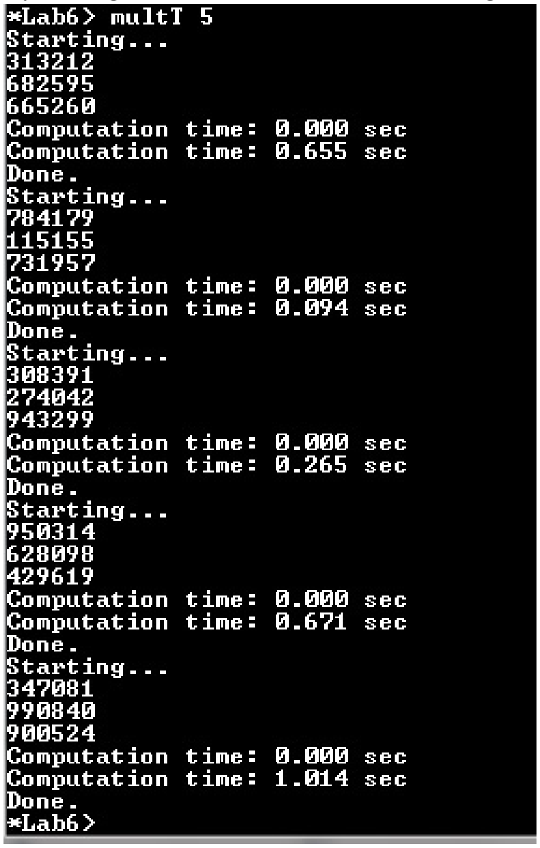
Exercise 2

```
-- EXERCISE 2
time :: IO t -> IO t
time a = do
    start <- getCPUTime
    v <- a
    end   <- getCPUTime
    let diff = (fromIntegral (end - start)) / (10^12)
    printf "Computation time: %0.3f sec\n" (diff :: Double)
    return v

timeEx x y m = do
    putStrLn "Starting..."
    putStrLn (show x)
    putStrLn (show y)
    putStrLn (show m)
    time $ exM1 x y m `seq` return ()
    time $ expM x y m `seq` return ()
    putStrLn "Done."

multT :: Integer -> IO ()
multT 0 = return ()
multT n = do
    x <- randomRIO (1000,1000000)
    y <- randomRIO (1000,1000000)
    m <- randomRIO (1000,1000000)
    a <- timeEx x y m
    b <- multT(n-1)
    return ()
```

By timing the two functions at calculating $rem(x, y)m$ the following results show up:



```
*Lab6> multT 5
Starting...
313212
682595
665260
Computation time: 0.000 sec
Computation time: 0.655 sec
Done.
Starting...
784179
115155
731957
Computation time: 0.000 sec
Computation time: 0.094 sec
Done.
Starting...
308391
274042
943299
Computation time: 0.000 sec
Computation time: 0.265 sec
Done.
Starting...
950314
628098
429619
Computation time: 0.000 sec
Computation time: 0.671 sec
Done.
Starting...
347081
990840
900524
Computation time: 0.000 sec
Computation time: 1.014 sec
Done.
*Lab6>
```

Where the first computation time is of exM1 and the second one of expM. exM1 is thus significanlty faster.

Exercise 4

```haskell
-- EXERCISE 4
-- shows all the composite numbers primeF checks as prime
testF :: Int -> IO [Integer]
testF k =
    filterM (primeF k) (take 50 composites)
-- Run multiple tests
testFMore :: Integer -> Int -> IO [[Integer]]
testFMore 0 _ = return []
testFMore n k = do
    c <- testF k
    d <- testFMore (n-1) k
    return $ filter (not . null) (c:d)
```

k = 1, k =2 and k = 3 give 4 as a prime number. When the value of k gets higher there are less fool primes found, this is because more different random numbers are chosen for a which lowers the probability a composite number is considered a prime.

Exercise 5

```haskell
-- EXERCISE 5
testCar :: Int -> IO [Integer]
testCar k =
    filterM (primeF k) (take 50 carmichael)
```

```
testMore1 :: Integer -> Int -> IO [[Integer]]
testMore1 0 _ = return []
testMore1 n k = do
    c <- testCar k
    d <- testMore1 (n-1) k
    return $ filter (not . null) (c:d)
```

Carmichael numbers almost always pass the Fermat's primality check. That was also shown in the testing, most of the numbers passed our test.
The Carmichael numbers are of the form $b^n \equiv b \pmod{n}$ for all integers $1 < b < n-1$. This is also how Fermat's little theorem define prime numbers ($a^{p-1} \equiv 1 \pmod{p}$.) . Because Fermat defines prime numbers in the same way Carmichael defines the Carmichael numbers, the carmichael numbers do satisfy the definition of a prime number used in Fermat's primality check. Although the carmichael numbers are not prime numbers but do satisfy Fermat's definition of a prime number, they pass the testing.

Exercise 6

```
-- EXERCISE 6
testMR :: Int -> IO [Integer]
testMR k =
    filterM (primeMR k) (take 50 carmichael)

test2More :: Integer -> Int -> IO [[Integer]]
test2More 0 _ = return []
test2More n k = do
    c <- testMR k
    d <- test2More (n-1) k
    return $ filter (not . null) (c:d)
```

Although some Carmichael numbers still pass the Miller-Rabin primality, these are significantly less than with Fermat's primality check. If we higher k, meaning that we check with more random *a's* we even find that Miller-Rabin doesn't consider any Carmichael numbers as prime numbers.

```
-- functions to show the difference in result of Fermat's primality and Miller-Rabin's primality check
lengthCar :: Integer -> Int ->  IO String
lengthCar n k= do
    f <- testMore1 n k
    return $ show (length f)

lengthMR :: Integer -> Int ->  IO String
lengthMR n k = do
    f <- test2More n k
    return $ show (length f)
```

```
*Lab6> lengthCar 500 10
"500"

*Lab6>lengthMR 500 10
"0"
```

Exercise 7

```
-- EXERCISE 7
--take a large prime, use miller rabin to check if 2^p -1 ook prime is (dan is het een mersenne getal)
multipleMersenne :: Integer -> Int -> IO [(Bool, Integer, Integer)]
multipleMersenne 0 _ = return []
multipleMersenne n k = do
    m <- mersenne k
    c <- multipleMersenne (n-1) k
    return $ (m : c)

mersenne :: Int -> IO (Bool, Integer, Integer)
mersenne k = do
    p <- randomPrime
    m <- primeMR k ((2^p) - 1)
```

```
    return $ (m,p,((2^p) - 1)) --

randomPrime = do
    b <- (randomRIO (1,100))
    return (primes !! b)
```

The numbers that give True for the first argument in the tuple are indeed known Mersenne numbers as can be found on `http://en.wikipedia.org/wiki/Mersenne_prime`.