

Dublin City University
School of Electronics Engineering
Assignment 2 Submission

<u>Student Name</u>	Dhruv Joshi
<u>Student ID</u>	20216445
<u>Programme</u>	Master's in Electronics and Computer Engineering
<u>Project Title</u>	Implementation and Assessment of PRNGs
<u>Module code</u>	EE501
<u>Project Due date</u>	29-04-2021
<u>Lecturer</u>	Mr. Xiaojun Wang

Objective

To gain an understanding of pseudorandom number generator (PRNGs) and their statistical properties

Description of Assignment

There are many different pseudorandom number generators (PRNG) for different purposes. This assignment is to implement two PRNGs of your choice, of which at least one should be cryptographically secure. The statistical properties of the implemented PRNGs, i.e. the ability to produce numbers that appear random, should be assessed using well-established metrics.

Programming Language used:

- HTML
- JAVASCRIPT

PRNG

Pseudorandom number generator

A pseudorandom number generator, also termed as a deterministic random bit generator (DRBG), is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequence of random numbers.

The PRNG-generated sequence is not fully random, because it is completely determined by a relatively small set of initials values, called the PRNG's seed (which may include truly random values)

Although sequences that are closer to truly random can be generated using hardware random number generators, pseudorandom number generators are important in practice for their speed in number generation and their reproducibility.

Implemented PRNG methods

Middle Square Algorithm

Here we start with a number called seed, lets assumes it's a 4 digit number 4321, as the algorithm name suggest ,we square it and extract the middle four digit **(4321) = 18671041**.

Now the next term in sequence **(6710) = 45024100**. Then we just repeat this process,

In case the square is less than 8 digit like 241=58081 in that we will pad it with leading zero's until its eight digit long **000** which makes it **00058081**. Starting with a four digit seed, the middle square algorithm generates a sequence of random-looking number between 0 and 9,999

For this task, I have write up a code by using the middle square algorithm in order to generate pseudo random number, please refer the below snippet of the code.

```
<script type="text/javascript">
  var digit = 4;
  seed = 1234;

  /** then using a function nextRand which will square the seed
  and covert that into a string store that value into a variable called n*/

  function nextRand() {
    var n = (seed * seed).toString();

    /* To make sure the digit lenght is not less that eight
    we have added n.lenth and where n will be padded with zeroes */
    while (n.length < digit * 2) {
      n = "0" + n;
    }

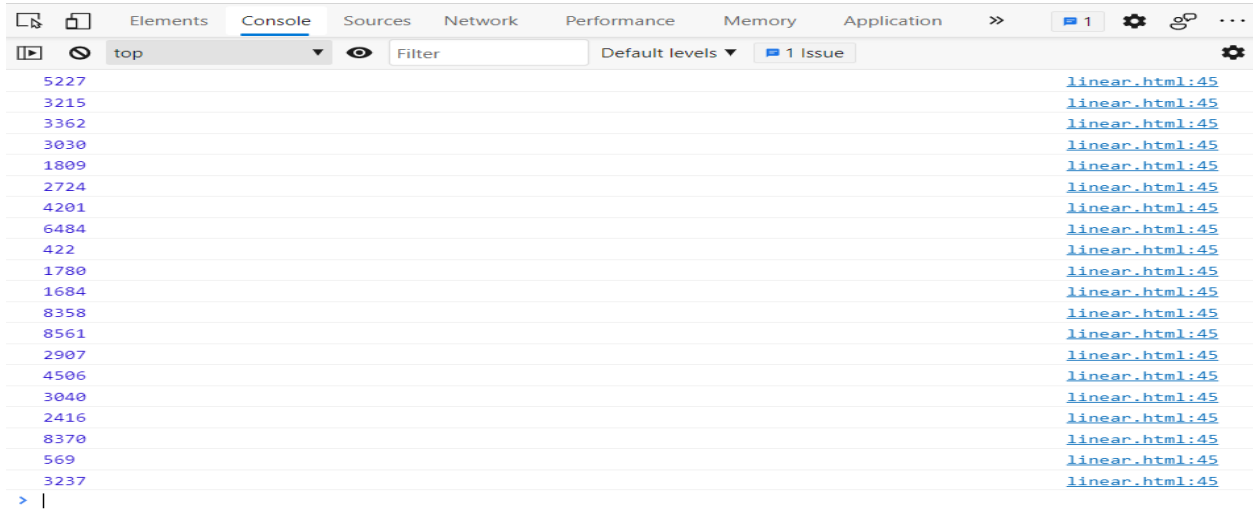
    /* to get the middle four digit we have performed string manipulation*/
    var start = Math.floor(digit / 2),
        end = start + digit;
    /** now the new seed will be n.substring and it
    will be a number so we have wrapped it parseInt*/
    seed = parseInt(n.substring(start, end));
    return seed;
  }

  /**in order to test this we are setting up a
  for loop for testing the next randomnumber 20 times*/
  for (var i = 0; i < 20; i++) {
    console.log(nextRand());
  }
}
```

Here as an output we have receive 20 random number refer the below snapshot

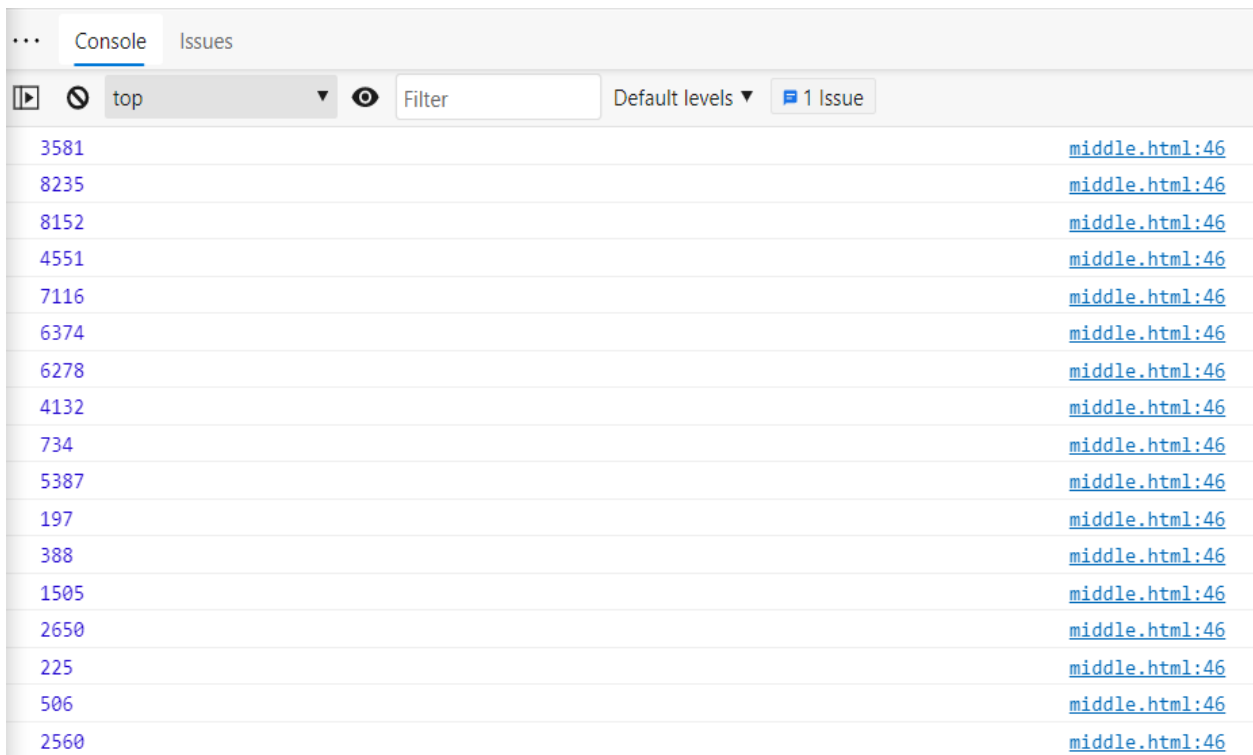
Test conducted for the random functionality of the code

Test 1 Using seed value 1234 and digit value 4







Random Number	Source
5227	linear.html:45
3215	linear.html:45
3362	linear.html:45
3030	linear.html:45
1809	linear.html:45
2724	linear.html:45
4201	linear.html:45
6484	linear.html:45
422	linear.html:45
1780	linear.html:45
1684	linear.html:45
8358	linear.html:45
8561	linear.html:45
2907	linear.html:45
4506	linear.html:45
3040	linear.html:45
2416	linear.html:45
8370	linear.html:45
569	linear.html:45
3237	linear.html:45

Test 2 Using seed value 4512 and digit value 4



Random Number	Source
3581	middle.html:46
8235	middle.html:46
8152	middle.html:46
4551	middle.html:46
7116	middle.html:46
6374	middle.html:46
6278	middle.html:46
4132	middle.html:46
734	middle.html:46
5387	middle.html:46
197	middle.html:46
388	middle.html:46
1505	middle.html:46
2650	middle.html:46
225	middle.html:46
506	middle.html:46
2560	middle.html:46

Test 3 Using seed value 4552 and digit value 5

... Console Issues	
  top  Filter Default levels  1 Issue	
20720	middle.html:46
29318	middle.html:46
59545	middle.html:46
45607	middle.html:46
79998	middle.html:46
99680	middle.html:46
36102	middle.html:46
3354	middle.html:46
11249	middle.html:46
26540	middle.html:46
4371	middle.html:46
19105	middle.html:46
65001	middle.html:46
25130	middle.html:46
31516	middle.html:46

Also I have added one more functionality to the code from we can check the period of the random number sequence generated

Below code shows the period = 10 when we want to generate 10 random numbers. Hence, all the number are unique.

```
var results = [];  
for (var i = 0; i < 10; i++) {  
    var rand = nextRand();  
    //      /*here we test if the results are already true  
    //      if so it will break out from the loop*/  
    if (results[rand]) {  
        break;  
    }  
    results[rand] = true;  
}  
console.log(i);
```

Output

```
10  
>
```

Below code shows the period = 56 when we want to generate 100 random numbers. Hence only 56 unique random number will be generated.

```
var results = [];  
for (var i = 0; i < 100; i++) {  
    var rand = nextRand();  
    //      /*here we test if the results are already true |  
    //      if so it will break out from the loop*/  
    if (results[rand]) {  
        break;  
    }  
    results[rand] = true;  
}  
console.log(i);
```

Output

```
10  
56  
>
```

Linear congruential generator.

Mersenne Twister: It is based on the much simpler linear congruential generator.

In this generator we have for inputs the modulus m , the multiplier a , the increment c , and the seed,

<u>MODULUS</u>	<u>MULTIPLIER</u>	<u>INCREMENT</u>	<u>SEED</u>
<u>m</u>	<u>a</u>	<u>c</u>	<u>Starting value</u>

The sequences starting value, here we goes with an example m equals 7829, a equals 378, c equals 2310 and **seed** is 4321

Multiply the seed by a added by c and then take the whole thing **mod** m .

$(Seed * a + c) \bmod m$

$(4321 * 378 + 2310) \bmod 7829 = 7216$,

Repeat this over and over in order to get the rest of the sequence

$$X_{n+1} = (aX_n + c) \bmod m$$

The number in the resulting sequence will be between 0 and 7828 which is m minus 1 (m-1)

For this task 2, I have write up a code by using the linear congruential algorithm in order to generate pseudo random number, please refer the below snippet of the code.

```
<script type="text/javascript">
/*In this task we are using four variable as the alogorith define
a,b,m and seed
for m we are using math.pow
as its function returns the base to the exponent power
also we change the value of seed and other variable to get different prn*/

var a = 12152712,
    c = 12345,
    m = Math.pow(2, 31),
    seed = 2589;
// for this function we implement seed with a basic formula of linear congruentail generator
// and return the seed value
function nextRand() {
    seed = (a * seed + c) % m;
    return seed;
}
// here we also update our randomfloat function
// the maximum value the float can return is under m

function nextRandFloat() {
    return nextRand() / m;
}
/**in order to test this we are setting up a
for loop for testing the next randomnumber 100 times**/
for (var i = 0; i < 100; i++) {
    console.log(nextRand());
}
</script>
```

Output

Test conducted for the random functionality of the code

Test 1 using value A=11035145,C= 12345,M=(2,31) &Seed= 1245

Console		Issues
⏮	top	Filter
853865982	linear.html:43	
1592267304	linear.html:43	
391914400	linear.html:43	
2027991257	linear.html:43	
1375828440	linear.html:43	
1752312016	linear.html:43	
673398664	linear.html:43	
416250881	linear.html:43	
762048066	linear.html:43	
1162925195	linear.html:43	
1550638876	linear.html:43	
1469890612	linear.html:43	
943118860	linear.html:43	
289829540	linear.html:43	

Test 2 using value A=12152712,C= 54321,M=(2,31) &Seed= 2589

Console Issues		X
top	Filter	Default levels 1 Issue
1398654617	linear.html:43	
2004855928	linear.html:43	
2052357104	linear.html:43	
789560240	linear.html:43	
1961744816	linear.html:43	
267372976	linear.html:43	
805783985	linear.html:43	
1841084728	linear.html:43	
2047508976	linear.html:43	
1247603632	linear.html:43	
499079600	linear.html:43	
364628401	linear.html:43	
958445881	linear.html:43	
1675882872	linear.html:43	
1719935984	linear.html:43	

Test 3 using value A=789456,C= 321654,M=(2,31) &Seed= 5858

Console Issues		X
top	Filter	Default levels 1 Issue
329987606	linear.html:43	
1601948758	linear.html:43	
2053808214	linear.html:43	
261283926	linear.html:43	
2064048214	linear.html:43	
1162272854	linear.html:43	
1497817174	linear.html:43	
93 424075350	linear.html:43	

Comparison between the implemented PRNG (Linear congruential generator and Middle Square Algorithm)

<u>Middle Square Algorithm</u>	<u>Linear congruential generator</u>
<p>To generate a pseudo random number using middle square method, a seed value is chosen (X0) or (N). The value is squared and the middle values of the squared value are returned as the random number. The above method is repeated to get a sequence.</p> <p align="center"><u>Used algorithm</u></p> <pre>var digit = 4; seed = 1234;</pre> <pre>function nextRand() { var n = (seed * seed).toString(); /* To make sure the digit lenght is not less than eight we have added n.length and where n will be padded with zeroes */ while (n.length < digit * 2) { n = "0" + n; } /* to get the middle four digit we have performed string manipulation*/ var start = Math.floor(digit / 2), end = start + digit; }</pre>	<p>Simple equation is used to generate a sequence of random numbers - $X_i = (a \times X_{i-1} + c) \bmod m$</p> <p>Seed = x0 a, c and m are defined by person.</p> <p>a - is called the constant multiplier; c - is the increment m - is the modulus</p> <p align="center"><u>Used algorithm</u></p> <pre>var a = 12152712, c = 12345, m = Math.pow(2, 31), seed = 2589; // for this function we implement seed // and return the seed value function nextRand() { seed = (a * seed + c) % m; return seed; }</pre>
<p align="center">Example</p> <p><u>1st iteration</u></p> <p>X0 = 945964 X square = 894847889296 Random number = 847889</p> <p><u>2nd iteration</u></p> <p>X1 = 847889 X square = 718915756321 Random Number = 915756</p>	<p align="center">Example</p> <p>a = 2175143 x0 = 3553 c = 10653 m = 1000000</p> <p>293732 114329 934700 172753 489332 85129 759100 61953 644932 335929 623500 671153 760532 866729 527900 353 836132 677529 472300 49553 871732 768329 456700 and so on...</p> <p><i>c shares no common factors with m</i></p>

The Middle square method is good for generating the Pseudo random number but up to some extent as while performing the algo on the code the test for iteration period of number repetition is small.	But while performing the test for repetition for the randomness of number, so here the period is bit longer as compared to middle square algorithm
Also the middle square is bit simple and not so secure and the randomness of the number can be identified, as the algorithm is simple and clear and the cryptanalyst can easily find the value of seed.	But in Linear Congruential the process of identifying the number is not so easy, here the algorithm used is bit secure as its takes various values as an input and uses other parameters to make the randomness of the number like $\text{Math.pow}(x,y)$ (This function return the base to the exponent power)
The execution speed of code for generating the Pseudo number is slow.	The execution speed of generating the Pseudo Random number is fast and
<ul style="list-style-type: none"> >Not a good method >Period very small >Random number generated is repeated a lot of times. >Loop indefinitely >Cycle to a previous number 	Better than the middle square method as they are fast, easily implemented by the computer hardware.
Things to keep in mind – If N is an odd number, add a leading zero. N must always be even for this method to work	Things to keep in mind – The basic rule is that c shares no common factors with m

Conclusion

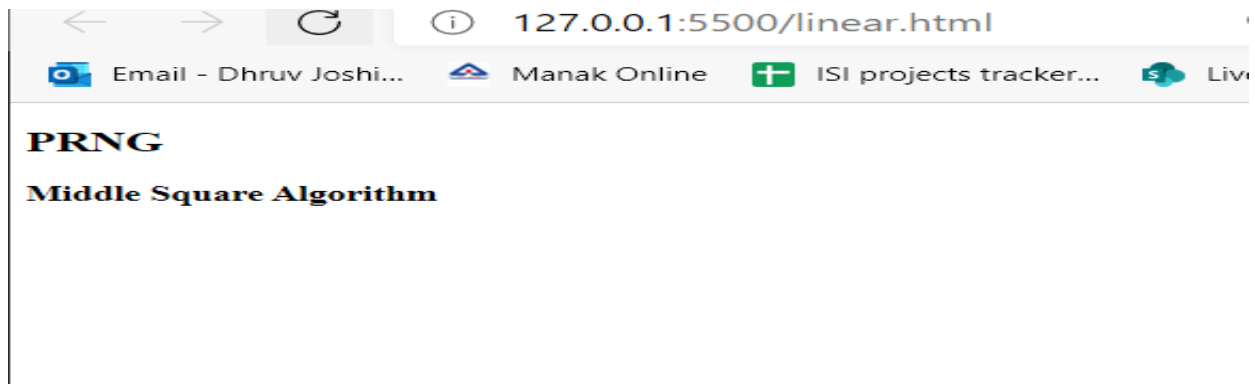
In this assignment we learn two different algorithm linear congruential generators and Middle Square Algorithm which is further used to implement Pseudo random number generation process, we also check the randomness of the number how often they get repeated by performing the randomness test from which we find out the iteration of the number and also we found that linear congruential algorithm is much safer and faster than middle square algorithm as it uses various parameters and values to generate a Pseudo random number.

Appendix

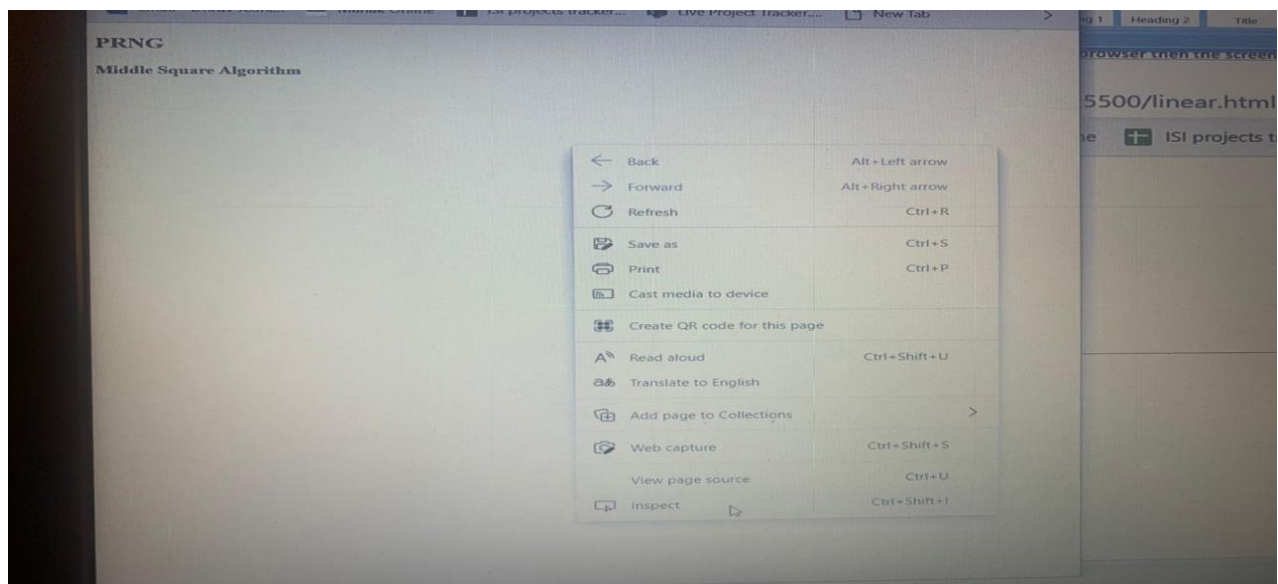
Steps to check/run the code,

For this assignment I have created two html files (linear.html and middle.html) which are enclosed with source code files written in Java script.

For checking the code, first open the html file on the browser (google, Microsoft), once the browser is open you will find a heading PRNG as shown on the below snippet



Once the browser is open the user needs to right click on the browser and inspect the page



Once the inspect tab is open, click on the **console** button and refresh the html page and once it's done the PRN will be shown on the console body. Please refer the below snippet.

