Documantation

True Random Number Generator

`TRNG.`**`BintoInt`**(*x*)

Inputs:

x - string of bytes

Output:

integer x

`TRNG.`**`TRNG`**(*m*, *k=1*)

Inputs:

m - intiger, 2^m will be upper range of generated values

k - integer, number of generated values

m * k < 20.001

Output:

list of x integers

`TRNG.`**`TRNGalg`**(*n*, *j*, *k*)

Inputs:

n - intiger, length of generated string of bytes

j - integer, length of cycle (PRNG will make j bytes from one seed)

k - integer, number of bytes in register

where j < 2*k, k < n

Output:

string, generated n bytes

Generator BBS

`BBS_gen.`**`BBSgen`**(*x*, *n*, *k*, *file_name*, *mode=2*)

Inputs:

x - intiger number, seed for generator

n - intiger, length of generating string of bytes

k - string, number added at the end of file name

file_name - string, name of creating file with result

mode - integer, 1 for bn - last byte, 2 for bn - xor all bytes, 3 for bn - compare number of bytes, 0 for receiving Xn

Output:

None, but saving generated n bytes to file

`BBS_gen.`**`BBSgen_body`**(*x*, *n*, *mode=1*)

Inputs:

x - intiger number, seed for generator

n - intiger, length of generating string of bytes

mode - integer, 1 for bn - last byte, 2 for bn - xor all bytes, 3 for bn - compare number of bytes, 0 for receiving Xn

Output:

string, generated n bytes

`BBS_gen.`**`BBSgen_elem`**(*x0*, *p*, *q*)

Inputs:

x0 - intiger number, seed for generator

p,q - integers, constant for generator

Output:

List of strings - next integer for generator and byte generated on 3 ways

Generator LFSR

`lfsr.`**`LFSR_body`**(*x0*, *k*)

Inputs:

x0 - string, seed for generator

k - integer, number of byte representation of seed

Output:

string - feedback from LFSR

`lfsr.`**`genLFSR`**(*x0*, *n*, *k*)

Inputs:

x0 - intiger number, seed for generator

k - integer, number of byte representation of seed

n - integer, length of final result

Output:

string - generated bytes from generator LFSR

`lfsr.`**`multixor`**(*x*, *n*, *c*)

Inputs:

x - string, seed for generator

c - string, variables of primitibe polynomials degree n

n - integer, length of seed

Output:

string - byte 0 or 1, result of operation XOR on seeds bytes

Generator based on mouse

`MouseGen.`**`MakePoints`**(*n*)

Inputs:

n - integer, number of creating points

Output:

list of coordinades of mouse coursor

`MouseGen.`**`MouseGen`**(*n*, *points*, *filename*)

Inputs:

n - integer, number of bytes in the end file

points - list of coordinates

filename - string, name of file, destination for generated bytes

Output:

None, list of bytes made from coordinates uploaded to file

`MouseGen.`**`MouseGenB`**(*n*, *points*, *filename*)

Inputs:

n - integer, number of bytes in the end file

points - list of coordinates

filename - string, name of file, destination for generated bytes

Output:

None, list of bytes made from polar coordinates uploaded to file

`MouseGen.`**`MouseGenC`**(*n*, *points*, *filename*)

Inputs:

n - integer, number of bytes in the end file

points - list of coordinates

filename - string, name of file, destination for generated bytes

Output:

None, list of bytes made from coordinates changed with map of chaos uploaded to file

`MouseGen.`**`PozycjaMyszy`**()

Inputs:

None

Output:

list of integers, coordinates of mouse coursor

`MouseGen.`**`SavePoints`**(*k*, *n*)

Inputs:

n - integer, number of creating points

k - integer, number of strings

Output:

None, but creates k files with points

`MouseGen.`**`TRMGBMultiple`**(*n*, *filename*, *m*)

Inputs:

n - integer, number of bytes in the end file

filename - string, prefix for name of file, destination for generated bytes

Output:

None, calls function MouseGenB m times

`MouseGen.`**`TRMGCMultiple`**(*n*, *filename*, *m*)

Inputs:

n - integer, number of bytes in the end file

filename - string, prefix for name of file, destination for generated bytes

Output:

None, calls function MouseGenC m times

`MouseGen.`**`TRMGMultiple`**(*n*, *filename*, *m*)

Inputs:

n - integer, number of bytes in the end file

filename - string, prefix for name of file, destination for generated bytes

Output:

None, calls function MouseGen m times

`MouseGen.`**`TRNG2MouseBon`**(*n*, *points*, *filename*, *j*)

Inputs:

n - integer, number of bytes in the end file

points - list of coordinates

filename - string, name of file, destination for generated bytes

j - integer, number of seed’s bytes

Output:

None, list of bytes made from LFSR with seed created from coordinates changed by chaos map uploaded to file

`MouseGen.`**`TRNGM2MouseBon`**(*n*, *filename*, *j*, *m*)

Inputs:

n - integer, number of bytes in the end file

filename - string, prefix for name of file, destination for generated bytes

j - integer, number of seed’s bytes

Output:

None, calls function TRNG2MouseBon m times

`MouseGen.`**`TRNGMLB`**(*n*, *points*, *filename*, *j*)

Inputs:

n - integer, number of bytes in the end file

points - list of coordinates

filename - string, name of file, destination for generated bytes

j - integer, number of seed’s bytes

Output:

None, list of bytes made from LFSR connected with BBS with seed created from coordinates, uploaded to file

`MouseGen.`**`TRNGMMLB`**(*n*, *filename*, *j*, *m*)

Inputs:

n - integer, number of bytes in the end file

filename - string, prefix for name of file, destination for generated bytes

j - integer, number of seed’s bytes

Output:

None, calls function TRNGMouseBon m times

`MouseGen.`**`TRNGMMouseBon`**(*n*, *filename*, *j*, *m*)

Inputs:

n - integer, number of bytes in the end file

filename - string, prefix for name of file, destination for generated bytes

j - integer, number of seed’s bytes

Output:

None, calls function TRNGMouseBon m times

`MouseGen.`**`TRNGMouseBon`**(*n*, *points*, *filename*, *j*)

Inputs:

n - integer, number of bytes in the end file

points - list of coordinates

filename - string, name of file, destination for generated bytes

j - integer, number of seed’s bytes

Output:

None, list of bytes made from LFSR with seed created from coordinates uploaded to file

`MouseGen.`**`TakePoints`**(*filename*)

Inputs:

filename - string, name of file with saved points

Output:

list of coordinates loaded from file

Generator based on microphone

`generator_mic.`**`generuj_plik`**(*nazwa_pliku*)

Inputs:

nazwa_pliku - string, name of creating file with result

Output:

None, but saving generated n bytes to file, function is related with arduino by serial port

Generator LFSR connected with BBS

`LFSRiBBS.`**`lfsribbs_body`**(*x0*, *n*, *k*)

Inputs:

x0 - intiger number, seed for generator

n - intiger, length of generating string of bytes k - integer, number of bytes in register

Output:

string, generated n bytes

Generator LCG

`pseudolosowy_afiniczny.`**`gen_LCG`**(*x0*, *n*, *file_name*)

Inputs:

x0 - intiger number, seed for generator

n - integer, len of output file

file_name - string, name of creating file with result

Output:

integer - lenght of creating file

`pseudolosowy_afiniczny.`**`gen_afi_body`**(*x0*, *a*, *b*, *M*, *n*)

Inputs:

x0 - intiger number, seed for generator

a, b, M - integers, constant for generator

n - integer, len of output file

Output:

string - n bits generated from generator LCG

`pseudolosowy_afiniczny.`**`more_byte`**(*x*)

Inputs:

x - intiger number

Output:

string - more frequent bit in binary representation of x

`pseudolosowy_afiniczny.`**`to_bin`**(*x*)

Inputs:

x - intiger number

Output:

string - the representation of x in binaries

`pseudolosowy_afiniczny.`**`xor_byte`**(*x*)

Inputs:

x - intiger number

Output:

string - score of operation xor made on bits of binary representation fo x

Tests for randomness

`testy.`**`Serie`**(*plik*)

Inputs:

plik - string of 20.000 bytes

Output:

dictionary where keys are lenhgts of runs and values are number of runs

`testy.`**`TestNajdluzszejSerii`**(*nazwa*)

Inputs:

plik - string of 20.000 bytes

Output:

Tuple of strings:

first element is boolean value “T” if test was succesfull “F” if test failed second element is value of the Test for the Longest Run

`testy.`**`TestPojedynczegoBitu`**(*plik*)

Inputs:

plik - string of 20.000 bytes

Output:

Tuple of strings:

first element is boolean value “T” if test was succesfull “F” if test failed second element is value of the Monobit Test `testy.`**`TestPokerowy`**(*plik*)

Inputs:

plik - string of 20.000 bytes

Output:

Tuple of strings:

first element is boolean value “T” if test was succesfull “F” if test failed second element is value of the Frequency Test within a Block

`testy.`**`TestSerii`**(*plik*)

Inputs:

plik - string of 20.000 bytes

Output:

Tuple of strings:

first element is boolean value “T” if test was succesfull “F” if test failed second element is value of the Runs Test

`testy.`**`generuj_wyniki`**(*plik_in*, *plik_out*, *m*)

Inputs:

plik_in - string, name of the file with generated bytes, only the prefix without sumple number

plik_out - string, name of final file, where will be save the results of tests

m - number of sumples

Output:

None

`testy2.`**`ApEntropyTest`**(*filein*, *n*, *m*)

Inputs:

n - integer, length of word

m - integer, length of block

filein - string of bytes

Output:

float, value of Entropy Test

`testy2.`**`Entropy`**(*word*, *n*, *m*)

Inputs:

n - integer, length of word

m - integer, length of block

word - string of bytes

Output:

folat, value of Entropy

`testy2.`**`SerialTest`**(*filein*, *n*, *m*)

Inputs:

n - integer, length of word

m - integer, length of block

filein - string of bytes

Output:

Tuple of two float values

`testy2.`**`blocks_count`**(*n*, *m*, *word*)

Inputs:

n - integer, length of word

m - integer, length of block

word - string of bytes

Output:

dictionary, where keys are possible blocks and values are number of this blocks in word

`testy2.`**`psi_calc`**(*n*, *m*, *word*)

Inputs:

n - integer, length of word

m - integer, length of block

word - string of bytes

Output:

float, value of psi function

Helpfull modules

`akceptacja_co_drugi.`**`akceptacja_co_drugi`**(*plik_in*, *plik_out*)

Inputs:

plik_in - string, name of the file with generated bytes, only the prefix without sumple number

plik_out - string, name of final file for saving new bytes

Output:

None, function call codrugizpliku() 10 times

`akceptacja_co_drugi.`**`codrugidane`**(*dane1*, *dane2*)

Inputs:

dane1 - string of 20000 bytes

dane2 - string of 20000 bytes

Output:

string of bytes made from dane1 and dane2

`akceptacja_co_drugi.`**`codrugizpliku`**(*input1*, *input2*, *output*)

Inputs:

input1 - string, name of the file with generated bytes

input2 - string, name of the file with generated bytes

output - string, name of the final file, after changes from chapter 6.1.1

Output:

None, function saves new string of bytes in output file

`modify_points.`**`biegunowy`**(*points*)

Inputs:

points - list of points [xi, yi]

Output:

list of polar variables [ai, bi]

`modify_points.`**`biegunowy_body`**(*point1*, *point2*)

Inputs:

points - list with 2 arguments [x, y]

Output:

polar variable [a, b]

`modify_points.`**`chaos_map`**(*points*)

Inputs:

points - list of lists with 2 arguments [xi, yi]

Output:

list of lists with 2 arguments [ai, bi] after chaotic mapping

`modify_points.`**`chaos_map_body`**(*point*)

Inputs:

points - list with 2 arguments [x, y]

Output:

list with 2 arguments [a, b] after chaotic mapping

`naprawa_danych.`**`napraw_plik`**()

Inputs:

None

Output:

None, function is repairing file from serial port to be usefull for RNG

`xorowanie.`**`xordane`**(*dane1*, *dane2*)

Inputs:

dane1 - string with 20000 bytes

dane2 - string with 20000 bytes

Output:

string with 20000 bytes made from dane1 and dane2 by using xor on bytes bi from both of strings

`xorowanie.`**`xorowanie`**(*plik_in*, *plik_out*)

Inputs:

plik_in - string, prefix for name of input file

plik_out - string, prefix for name of input file

Output:

None

`xorowanie.`**`xorpliki`**(*input1*, *input2*, *output*)

Inputs:

input1 - string, name of input file

input2 - string, name of input file

output - string, name of output file

Output:

None, but saving to output file string with 20000 bytes made from dane1 and dane2 by using xor on bytes bi from both of strings