J. Dichter **Program #2**
CS 501 **University of Bridgeport**

Instructions: You will write a program to generate a series of pseudo random numbers, using a linear feedback shift register method. You will start with an arbitrary number (not zero) x with a representation in n bits. If x = 15 and n = 4, then x = 1111. Then you will select a pair of bits, say bits 0 and 1, take their *"xor"* producing a new bit - in this case a 0. The number x will then shift to the right, with the new bit becoming the new high bit - in this case 0111. The process continues in this manner to produce a random set of bits. With four bits, we can only have up to 15 unique transitions before repetitions begin, or 2^n - 1. With n = 32, the number of random bits has a much greater potential. The numbers actually produced by the "shifting" cannot be considered "random", since all but one of the bits overlap from each succeeding number. However, if we want to produce, say 100 6-bit numbers, then we need to shift the bits into an 6-bit register. We can then save its "value" in an array of 100 integers. Then we should produce the output which will display the distribution. Random numbers should follow a uniform distribution. Using a 32-bit generator does not guarantee 2^n - 1 random bits. For good results try using bit combinations 0/4, 0/7, 0/25, 0,26, 0/29.

Example:
11001**1**1**0** -> *1*1100**1**1**1** -> *0*1110**0**1**1** -> *1*0111**0**0**1** -> *1*1011**1**0**0** ->
*1*1101**1**1**0** -> *1*1110**1**1**1** -> *0*1111**0**1**1** -> *1*0111**1**0**1** -> *0*1011**1**1**0** -> etc

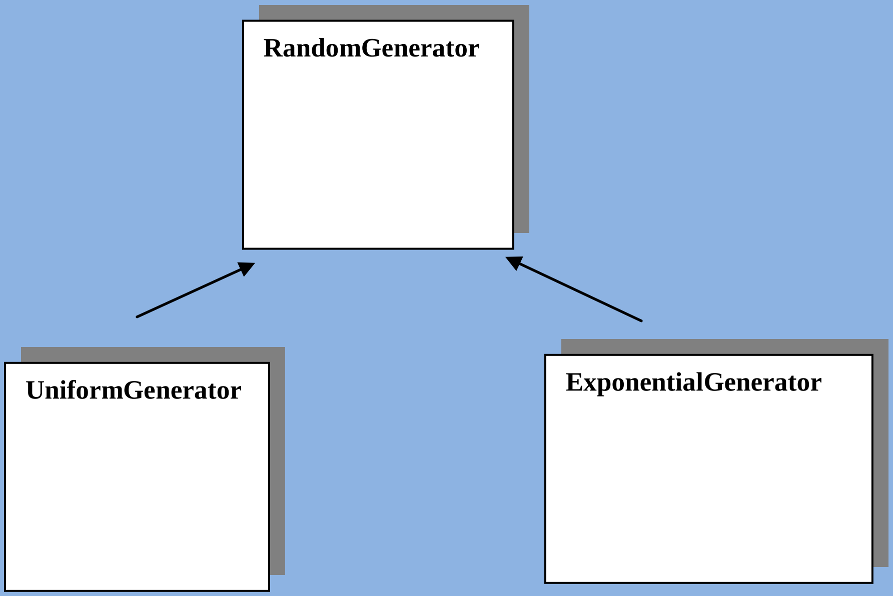
Three random 4-bit numbers produced: 11, 13, 4, ... Try "tapping" different bit combinations, as well as the ones I suggested. To verify that the set of numbers you generated are really pseudo random, you will test them using the statistical (Chi-Square) χ^2 test. You will generate 10,000 numbers in the 0-63 range. Then N = 10,000 and r = 64. You will then compute χ^2 by the formula

$$\chi^2 = \frac{\sum_{0 \le i < r} (f_i - N/r)^2}{N/r}$$

If χ^2 is in the range of $r \pm 2\sqrt{r}$, we conclude that distribution is indeed random. Otherwise it may not be. In your program, you will implement a random number generator module. Then the program will request 10,000 random numbers in the range of 0-25. It will then apply the test to see if the numbers are random. You should apply the test a number of times because it has a 1/10 chance of showing failure even when the random distribution is successfully generated.

You will implement the random number generator as a `RandomGenerator` C++ class. The constructor will have to initialize the seed value. Subsequent calls to the random( ) method with integer parameter *range* will return a random value in the range between 0 to *range*-1 inclusive. The tapping bits should be set by default (in the constructor).

You will create a class called `RandomGenerator`. You will use inheritance to create two subclasses, `UniformGenerator` and `ExponentialGenerator`. The 1st will generate random integers in a range from 0 to *n* –1. The 2nd will generate random real numbers in an exponential distribution with parameter lambda. The class hierarchy will look as follows:



Be sure that your design factors the common behaviors of all generators into the *superclass*, `RandomGenerator`. After running your program several times you will take the output from the file you generated, and put it though a test. The exponential numbers should follow the exponential distribution with parameter lambda, λ. You should take the numbers, and graph them in a spreadsheet, such as Excel. The shape of your distribution will reveal the quality of your generated numbers. A histogram may be a good way to visualize the numbers. It is well known that the mean of the exponential distribution is 1/ λ. For example, if λ = 0.5, then your mean should be quite close to 1/ 0.5 = 2. When you print out your mean, you should get close to 2.0000, but not exactly 2.0000.

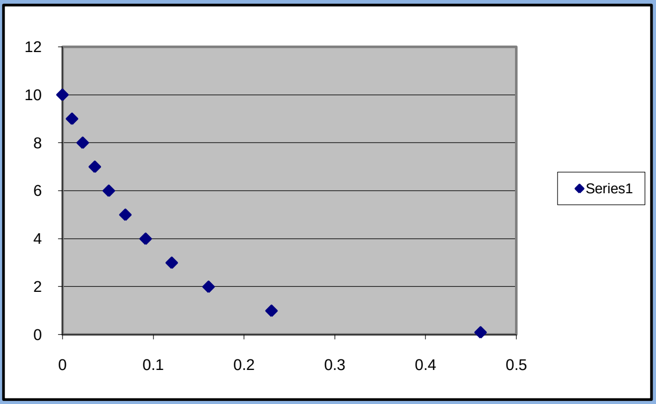
The uniform numbers will tend to have a similar number of occurrences for all numbers in the range. You should use a spreadsheet again to show your result. In addition, you should also include the χ^2 test along with the uniform numbers.

The for the `UniformGenerator` does not need any parameters. The `ExponentialGenerator` needs the lambda (λ) parameter.

Lambda 10

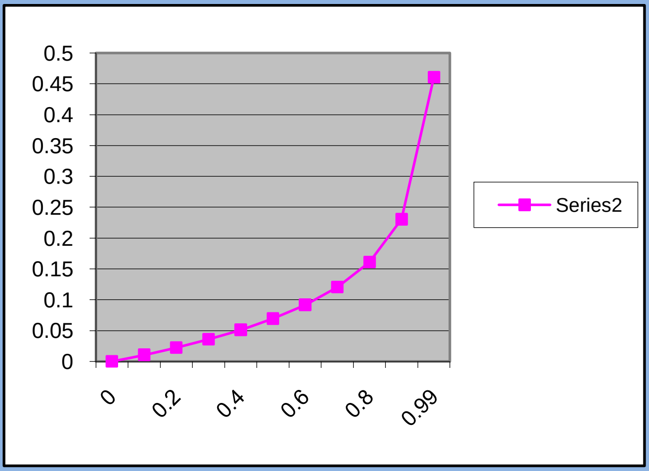
density function λ *pow(e, - λt)

t	**f(t)**
0	10
0.010536	9
0.022314	8
0.035667	7
0.051083	6
0.069315	5
0.091629	4
0.120397	3
0.160944	2
0.230259	1
0.460517	0.1



distribution function - (1 / λ) ln (1 - z)

z	**t**
0	0
0.1	0.010536
0.2	0.022314
0.3	0.035667
0.4	0.051083
0.5	0.069315
0.6	0.091629
0.7	0.120397
0.8	0.160944
0.9	0.230259
0.99	0.460517


Series2