Advanced Computer Programing - 159.732

Assignment 2

Tim McMullen - 06222757

This assignment required the creation of a random number generator(RNG). To do this I created a program that was able to call external functions as necessary to get the required results. These functions ranged in functionality, as some were used to create random numbers while others were used to save the current state of the random numbers, or manipulate the data to return a new value.

The program starts by seeding some values by using a Linear Congruential Generator, these values are then stored in a global array to allow for an easy method of passing current numbers and allows for another functions to use to the same numbers, and edit or replace them as needed. By using these functions the majority of the code can be change by the user very easily, to produce a different resulting RNG, or transforming the resulting data's distribution, such as the use of the Box-Muller algorithm. The data has been stored as a unsigned long, to allow for a larger range of values generated. A Complementary multiply-with-carry RNG has also been implemented as this is faster and has a larger period than that of other lag table based methods. The Complementary multiply-with-carry is currently the defult RNG within this program but by simply changing the function call to seed() you are able to switch to a Linear Congruential Generator. I did try to implement a lagged Fibonacci generator, but was able to get the required results though it works a little it is able to get stuck in a zero state in which it takes awhile to get out of.

To save or load the current of a generator we simply need to save the global array storing all its values, or load the file to replace its current values with those that were previously saved, this can be done with the save() load() functions.

To implement a new RNG within this program one must only create a new function and have it access the global variables, and return an appropriate unsigned number, this allows for current code to be kept and not needed to be replaced or redundant code created.

This current implementation is unable to calculate the standard deviation, as numbers are not store for a large period of time, but this could be done by implementing a table to store all generated values, also it is lacking a routine to calculate the binomial distribution of the numbers generated. I had hoped to test these RNGs using the diehard tests, but was unable to correctly implement them

The algorithms used were picked for speed, yet the Complementary multiply-with-carry also has a larger period than a Mersenne Twistor, with equally good results, The numbers used within these RNG were taken from already implemented RNGs to improve the quality of the numbers generated.

sources:

http://www.taygeta.com/random/gaussian.html - box mill implementation

http://en.wikipedia.org/wiki/Mersenne_Twister - Complementary multiply-with-carry theory

http://i.cs.hku.hk/~diehard/cdrom/ - diehard tests