**Assignment #1  
Due September 19th**

You should write all your programs on the Unix system that you used for the lab, or on your own Unix system. You'll have more flexibility for your platform in later assignments! Turn in your C++ source code files via Blackboard (this means you will probably want to download the files to your own computer). Please see the submission instructions for more details.

1) **Benford's Law** **- 30 pts**  
Given a collection of naturally occurring numbers - for example, the length of rivers around the world, enrollment at UAA over the years, the number of votes for a candidate by precinct – we might expect these numbers to have nothing in common.   
  
But what if you looked at the distribution of the leading digit of these numbers? In other words, count the number of times 1 is the leading digit, 2 is the leading digit, etc. A natural assumption is that all of these digits are equally likely so there should be about a 11% chance to see any one value from 1-9 as the leading digit. What we find instead is quite remarkable. 1 is the leading digit about 30% of the time, and the probability drops to around 5% for the digit 9. This distribution holds for all of these examples despite the disparate sources and is called [Benford's Law](http://en.wikipedia.org/wiki/Benford's_law). One application of this phenomenon is verification of voting records - if the number of votes by precinct doesn't match the expected distribution then that raises questions about vote tampering.   
  
The file enrollments.txt (in ~sjneumayer/CSCE\_A211 on **ruminant.duckdns.org**) on contains 3295 UAA enrollments by section. Write a C++ program that reads 3295 numbers from the **console** (i.e., you would normally type in the numbers) and outputs how many of those numbers start with 1, how many start with 2, etc. up to 9. You might want to look ahead at arrays to make your job a little easier. Run your program in Linux and use [**I/O redirection**](http://linuxcommand.org/lc3_lts0070.php) **(**<http://linuxcommand.org/lc3_lts0070.php>) (i.e. program\_name < file\_name) to read the numbers from the file instead of you typing in all 3295 numbers. **Your program should not open the text file and read it directly! To get all the points you must use I/O redirection from the linux shell.** The results should generally match Benford's law.

2) **Chuck-a-luck** **- 30 pts**

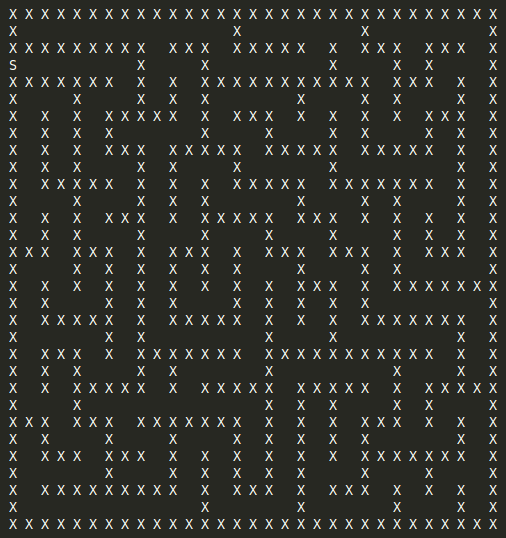
Chuck-a-luck is a simple dice game for gamblers. You can play it in some Vegas casinos. To play, the gambler first places a wager, *W*. The gambler then picks any number from 1-6. Let *n* represent the gambler's number. Next, the gamber rolls three six-sided dice. If none of the rolls equals *n* then the gambler loses *W*. If one of the rolls equals *n* then the gambler wins wager *W* back (even money, 1-1). If two of the rolls equals *n* then the gambler wins twice the wager *W* (2-1). If all three of the rolls equals *n* then the gamber wins three times *W* (3-1).

Write a program that implements a C++ version of Chuck-a-luck. It should output appropriate messages so the player knows what has happened. The player should be prompted to input a wager, select a number, and to initiate rolling the dice. The gambler should start with $100. The gambler should be allowed to keep playing until he or she runs out of money (don't allow a wager that is more than the amount of money owned) or the gambler decides to quit playing. For the selected number that should be from 1-6, your program should only allow the gambler to choose a valid number. For example, if 7 or 0 is entered then the gambler should be prompted to enter the number again.

Additional requirement: Your solution must use at least one programmer-defined function in a meaningful way. You are welcome to write more functions if you wish.

3) **Maze Generation and Maze Solving** - **20 pts**

There is no code to write for this problem. Consider a maze that is represented by a 2-D array. A sample maze is shown in the image below:



Walls are shown with an X, spaces with a blank, and the exit is S.

1. Given a random start location, think of an algorithm that does not use recursion that would find its way to the exit S. Your algorithm should be able to run on an arbitrary maze, and not be specific to the sample given above. Can your algorithm avoid going in circles? Describe your algorithm.

My answer: Given the start location I would make my algorithm go to any spaces that have a blanks spot. Along the way, I would make a 2D array that checks coordinates and marks if a spot has been visited to prevent circles. If a spot is marked in an X or has been visited before it would stop continuing that line. Finally the program will terminate when it reaches the exit ,‘S’,

1. Think of a way to have a computer program generate a random maze. What are important criteria that a good maze should have? Describe how your algorithm could possibly meet the criteria.

My answer: There are a few criteria a good maze should have.

* + 1. The boundary of the wall is clear is understandable
    2. There is a path from exit to the beginning point
    3. The path from exit to the beginning is not a straight shot
    4. There are an apt number of dead ends to aid in complexity
    5. Finally, it has to be random

My algorithm would start by making the grid filled with x’s and choice where the start end endpoints are. Then I would make other starter points in the maze and make it go from a false start to a false end. In doing so, making those paths random to aid in its random generation. When those are done, I generate the real path making it go through the already clear path but allowing it to break walls if there is no other way.

4) **Reference parameters - 20 pts**

Write a program that reads an integer from 1-99 that represents some amount of change to dispense. The program should calculate the minimum number of coins in terms of quarters, dimes, nickels, and pennies that adds up to the amount of change. Write a void function that takes as input:

* An int **passed by value** that represents the amount of change to dispense
* An int **passed by reference** that returns the number of quarters needed
* An int **passed by reference** that returns the number of dimes needed
* An int **passed by reference** that returns the number of nickels needed
* An int **passed by reference** that returns the number of pennies needed

Pay special attention to the bolded items above. You must use reference parameters within a single function for the quarters, dimes, nickels, and pennies. The main function **MUST use this void function to calculate the number of quarters, dimes, nickels, and pennies to dispense as change**. Write test code in the main function that calls your function with at least two test cases and outputs how many of each coin is needed to add up to the amount of change.