CSC111 Introduction to Computer Science Fall 2017
Lab 3 – Happy Birthday

For purposes of this lab, we make the simplifying assumption that all years have 365 days. I.e., we will not (yet) take leap years into account.

You may have heard of a popular party trivia question: "Who has the same birthday ?" Here "birthday" means which day of the year, not including the exact year of their birth. If you are having a large gathering, i.e., with more than 365 people,[1] it is obviously a certainty that there must be at least two people in the group with with the same birthday.[2]

But, what if you have fewer people at your big bash, say 20 or 30 ? If you ask most people, they would guess that it is unlikely that two people out of 30 have the same birthday. They would be wrong. It takes surprisingly few people in a group to have a high probability that two or more people have the same birthday. The goal of this lab is to estimate the probabilities that in a group of N people, two or more people will have the same birthday, where $2 \leq N \leq 50$. We will use simulation to estimate the probabilities.

Input: None.

Output: A table of the number of people (from 2 to 50) and the estimated probability that a group of that size has at least two people with the same birthday.

Implementation Guidelines:

- We use the numbers { 0, 1, 2, 3, ..., 364 } to identify the 365 days of the year.
- Let N denote the number of people in our group. To simulate N birthdays, we can generate N random numbers in the range 0 to 364. As we generate each simulated birthday, we need an efficient way to recognize if a previously simulated birthday matches the current one. An efficient way to do this is outlined in pseudocode below:

```
1. Let A denote an array of integers, previouly allocated to size 365.
   A[k] is a count of the number of people who have their birthday on day k.

2. Set every entry in the array to zero.  We start our counts with zero.

3. have_match = false ;  // No birthdays on the same day found yet.

4. loop b = 1, 2, 3, ... N  {  //  Generate N birthdays.
      Let k = a random number in the range 0 to 364 ;
      A[k] = A[k] + 1 ;  //  We count this person with birthday k.
      if ( A[k] > 1 ) {
          have_match = true ;
      }

5. The boolean variable 'have_match' is true if and only if
   two people in this group have the same birthdays.
```

[1] Assume no one is born on February 29.

[2] This logical proof technique is often called "pigeon hole principle".

The observant reader will notice that once a match is found in step 4, there is no need to continue iterating the loop. Once `have_match` is set to true, it can never be set back to false within the loop. A good way to implement the search for a match is to use a `while` loop instead. A compound condition is needed to express that the loop control variable `b` is within range, and `have_match` is still false.

- For each value of N in the set $\{2, 3, 4, ...50\}$, repeat the simulated group experiment 100,000 times. Let m denote the number of times out of the 100,000 trials that (at least) two people are found with the same birthday. Then our estimate of the probability is given by:

$$P(\text{ two or more persons with the same birthday }) \approx \frac{m}{100000}$$

Coding Guidelines:

- Include your name in a comment section at the top.
- Name your source code **birthday.java**.
- Name your class similarly, i.e., **class birthday**.
- Use proper indentation.
- Use named constants.
- Include comments describing the role of each variable.

Program Organization:

Your should organize your program into several methods:

- Write a method (named `one_group()`). The input is the number of people, N. Perform one simulation of a group of N people, and return **true** if two or more people are found to have the same (simulated) birthday. Return **false** otherwise.
- Write a method (named `one_probability()`). The input is the number of people, N. Run the method `one_group()` 100,000 times and count the number of **true** outcomes. Return the estimated probability (use type double).
- Write the main method to call `one_probability()` for each of the group sizes $2, 3, 4, ..., 50$. At each iteration print the group size and the estimated probability.

Running Your Program

```
gottlieb% javac birthday.java
gottlieb% java birthday > probs
```

We will use a program named "octave" to draw a plot for us, showing the relationship between group size and the probability that at least two people have the same birthday. Save your plot to a file name "probs.png".

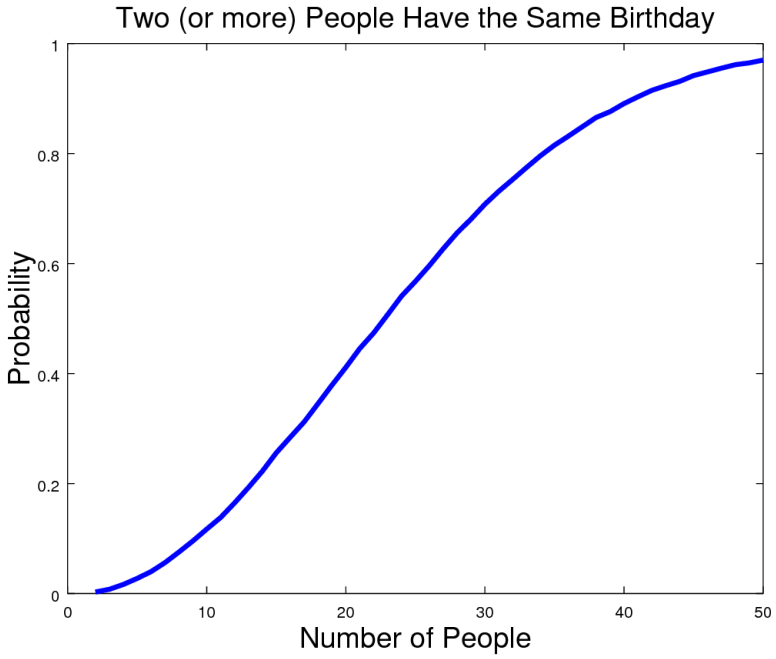
The octave commands you need are shown below:

```
load probs
npeople = probs(:,1) ;
pr = probs(:,2) ;
plot(npeople, pr, "linewidth", 3  ) ;
title("Two (or more) People Have the Same Birthday", "fontsize", 18) ;
xlabel("Number of People", "fontsize", 18) ;
ylabel("Probability", "fontsize", 18) ;
print("probs.png")
```

Expected Output: (shown in three columns for brevity)

2	0.0026	19	0.3657	36	0.8307
3	0.0095	20	0.407	37	0.850
4	0.0149	21	0.4433	38	0.8652
5	0.0264	22	0.4689	39	0.883
6	0.0397	23	0.5059	40	0.8884
7	0.0557	24	0.5363	41	0.9059
8	0.0732	25	0.5724	42	0.9137
9	0.0946	26	0.5988	43	0.923
10	0.1148	27	0.6283	44	0.9362
11	0.1441	28	0.6675	45	0.9458
12	0.166	29	0.6819	46	0.9489
13	0.1984	30	0.7021	47	0.9537
14	0.2192	31	0.7295	48	0.959
15	0.2519	32	0.7621	49	0.9659
16	0.2787	33	0.7763	50	0.9716
17	0.3062	34	0.794		
18	0.3515	35	0.8186		

Plot:



Turn In:

Question: What's the minimum number of people in a group to have at least a 50% chance that two (or more) people have the same birthday ? Are you surprised by the answer ?

Save all your work in a directory named "Lab3". Change to your home directory (the parent directory of "Lab3"), and create a file named "lab3.tar" using the command:

```
tar cf lab3.tar Lab3
```

Use `sftp` to upload the file "lab3.tar" to your account on telesto.

Birthday Party Animal