CS202 -- Lab 6

- **CS202 -- Data Structures and Algorithms I**
- **Fall, 2021**
- **James S. Plank**
- **This file: http://web.eecs.utk.edu/~jplank/plank/classes/cs202/Labs/Lab6**
- **Lab Directory: /home/jplank/cs202/Labs/Lab6**

To start this lab

Do the following:

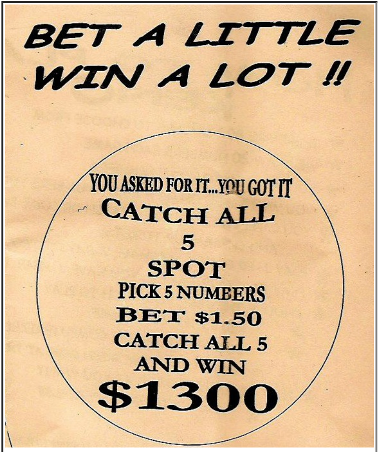
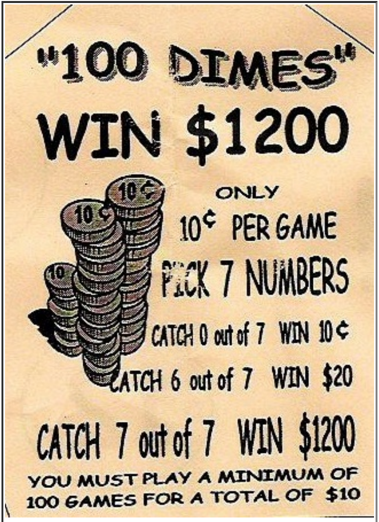
```
UNIX> cp -r /home/jplank/cs202/Labs/Lab6/src .
UNIX> cp -r /home/jplank/cs202/Labs/Lab6/include .
UNIX> cp /home/jplank/cs202/Labs/Lab6/makefile .
UNIX> mkdir obj
UNIX> mkdir bin
```

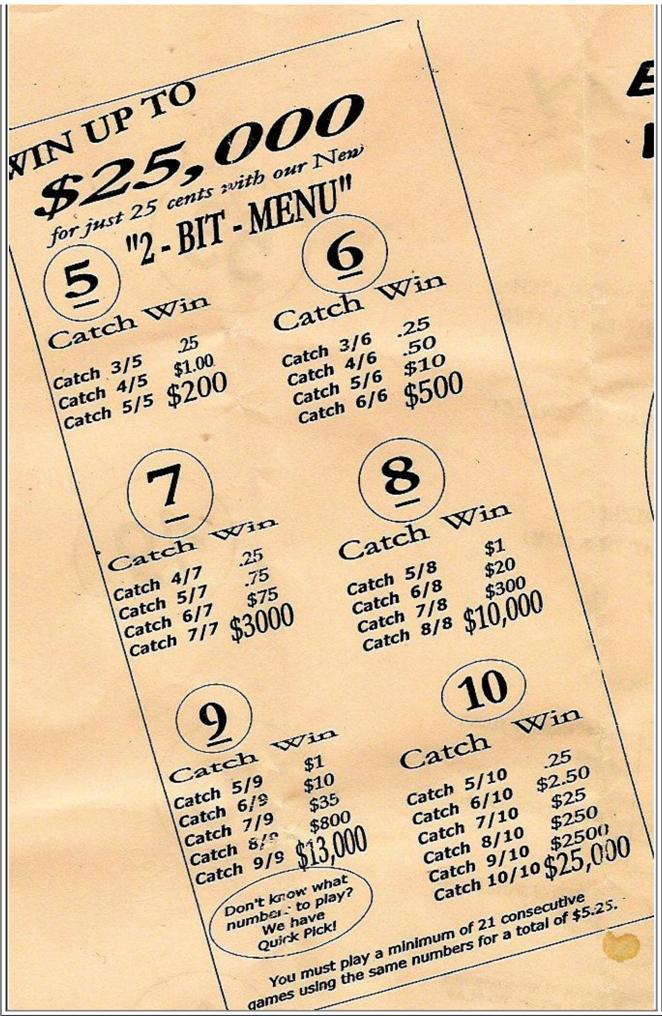
Your job will be to write two programs: **src/fraction.cpp**, which implements the Fraction class, and **src/keno.cpp**, which is described below.

The story

You are in Vegas with your wife and her family. On morning number 1, they all want to go to the "Venetian" hotel and casino to ride the Gondolas. You have the choice of going or staying. Hmmm, let's see. You've already overpaid once in your life to ride a real gondola in Venice. It's something like 500 bazillion degrees in Vegas, and the gondola rides are in a big concrete swimming pool. You thank the family, but send them on their merry way.

You wander around the casino, and see this game called *Keno*. It's a bit like a lottery. There are 80 balls numbered 1 through 80, and they will pick 20 of them randomly. They have a catchy little flier about all the Keno bets you can make:

	 BET A LITTLE WIN A LOT !! YOU ASKED FOR IT...YOU GOT IT CATCH ALL 5 SPOT PICK 5 NUMBERS BET $1.50 CATCH ALL 5 AND WIN $1300	 "100 DIMES" WIN $1200 ONLY 10¢ PER GAME PICK 7 NUMBERS CATCH 0 out of 7 WIN 10¢ CATCH 6 out of 7 WIN $20 CATCH 7 out of 7 WIN $1200 YOU MUST PLAY A MINIMUM OF 100 GAMES FOR A TOTAL OF $10



WIN UP TO
$25,000
for just 25 cents with our New
"2-BIT-MENU"

5

Catch	Win
Catch 3/5	.25
Catch 4/5	$1.00
Catch 5/5	$200

6

Catch	Win
Catch 3/6	.25
Catch 4/6	.50
Catch 5/6	$10
Catch 6/6	$500

7

Catch	Win
Catch 4/7	.25
Catch 5/7	.75
Catch 6/7	$75
Catch 7/7	$3000

8

Catch	Win
Catch 5/8	$1
Catch 6/8	$20
Catch 7/8	$300
Catch 8/8	$10,000

9

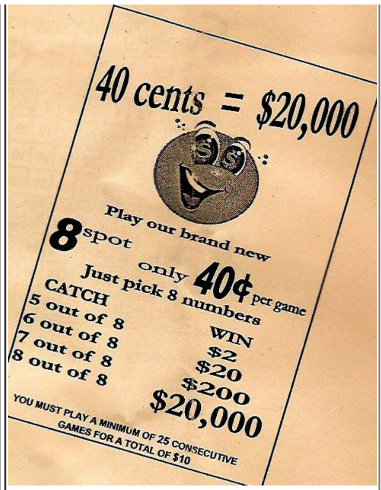
Catch	Win
Catch 5/9	$1
Catch 6/9	$10
Catch 7/9	$35
Catch 8/9	$800
Catch 9/9	$13,000

10

Catch	Win
Catch 5/10	.25
Catch 6/10	$2.50
Catch 7/10	$25
Catch 8/10	$250
Catch 9/10	$2500
Catch 10/10	$25,000

Don't know what numbers to play? We have Quick Pick!

You must play a minimum of 21 consecutive games using the same numbers for a total of $5.25.



40 cents = $20,000

Play our brand new 8 spot only 40¢ per game

Just pick 8 numbers

CATCH	WIN
5 out of 8	$2
6 out of 8	$20
7 out of 8	$200
8 out of 8	$20,000

YOU MUST PLAY A MINIMUM OF 25 CONSECUTIVE GAMES FOR A TOTAL OF $10

The whole flier is here.

Now, we're talking entertainment! You know there's no way that these tempting little bets are going to make you money in the long run, but have a little mathematical problem to solve, and that's better than gambling! Your job is to figure out which of these bets is the best -- in other words, which one will lose you the least amount of money in the long run.

Let's analyze the bet sheet. On the "2-Bit Menu" you will pay a quarter per game, which you will not get back. Let's say you choose the "5 catch win." This means that you will pick five numbers. They will pick 20. If exactly three of your five numbers are in their twenty, they pay you a quarter (i.e. you get your money back). If exactly four of your five are in their twenty, then they pay you a dollar. And if all five are your numbers are in their twenty, you win $200!!!!! Whoo-hoo!!!

So, suppose you want to calculate what the average return on your investment will be. The probability of matching exactly three balls is 0.0839351. The probability of matching exactly four balls is 0.0120923. And the probability of matching exactly five is 0.000644925. I'll show you how to calculate those later. So, to calculate your return:

- You always lose your initial 0.25
- 8.39351 percent of the time, you win back 0.25.
- 1.20923 percent of the time, you win back a dollar
- 0.0644925 percent of the time, you win $200.
- Therefore, your return is -0.25 + 0.0839351 * (0.25) + 0.0120923 * (1) + 0.000644925 * (200). That is -0.087939, so you lose roughly nine cents on every 25 cent bet.
- If you care to normalize that to the dollar, you normalized return is -0.087939/.25 = -0.35 per bet. That's one lousy bet. At least it only costs a quarter (although the fine print says you have to play 21 times....)

Onto the lab. In mathematics, a binomial coefficient *binom(n,k)* is defined to equal (n!)/((k!)(n-k)!). It represents the number of ways to choose *k* distinct items from a collection of *n*. Therefore, the total number of ways that you can choose 20 balls from 80 is *binom(80,20) = (80!)/(20!60!)*.

In Keno, suppose you pick *p* balls. Then the number of ways to match exactly *c* of those *p* in the twenty randomly chosen balls is:

*binom(80-p,20-c) * binom(p,c)*

So, the probability of you matching exactly exactly *c* of *p* is:

*binom(80-p,20-c) * binom(p,c) / binom(80,20)*

Let's take a concrete example. If we choose five balls and want to match exactly three, that's: *binom(75,17) * binom(5,3) / binom(80,20)*. Which equals:

$$\frac{75!\,5!\,20!\,60!}{17!\,58!\,3!\,2!\,80!}$$

Fortunately, we can cancel quite a few of these terms. For example, (75!)/(80!) = 1/(80*79*78*77*76). And (5!)/(3!) = 5*4. We can keep cancelling until we get:

$$\frac{5*4*20*19*18*60*59}{2*80*79*78*77*76}$$

This equals 0.0839351.

(For the true nerds, yes, we can do a prime factorization and cancel many more terms, but for the purposes of this lab, we'll just do simple cancellations).

Your Job: #1: fraction.cpp

You are to write two programs: **src/fraction.cpp** and **src/keno.cpp**.

The first implements a class called a "fraction", defined in **include/fraction.hpp**. Please read the comments in the header file for description:

```
#pragma once
#include <set>

/* The Fraction class manages a fraction, where the numerator and the denominator
   are both products of positive integers.  Internally, you will represent the fraction
   as two multisets -- one for the numerator and one for the denominator.  You want to
   make sure that the same number does not appear in both the numerator and denominator.
   When that happens, you should delete the number from both the numerator and the
   denominator.

   You manipulate the product with the first eight methods.  Print() prints the
   fraction and Calculate() calculates the fraction as a double. See the method
   descriptions for more information.

   For the methods that return a bool, return true when the operation is successful,
   and false if the parameters are bad. */

class Fraction {
  public:
    void Clear();                      // Clear both the numerator and denominator

    bool Multiply_Number(int n);       // Add a number to the numerator
    bool Divide_Number(int n);         // Add a number to the denominator

    bool Multiply_Factorial(int n);    // Add the numbers 2 through n to the numerator
    bool Divide_Factorial(int n);      // Add the numbers 2 through n to the denominator

    bool Multiply_Binom(int n, int k);  // Effect multiplying by n-choose-k
    bool Divide_Binom(int n, int k);    // Effect dividing by n-choose-k

    void Invert();                      // Swap the numerator and denominator

    void Print() const;                 // Print the equation for the fraction (see the lab writeup)

    double Calculate_Product() const;   // Calculate the product as a double.

  protected:
    std::multiset <int> numerator;
    std::multiset <int> denominator;
};
```

In the various calls, **n** must be a positive integer, and **k** must be greater than or equal to zero (anything choose 0 is equal to one). If any of these are bad, the various calls should return **false**.

The only subtle method is **Print()**. The way this works is to print the numerator as a bunch of products, and then the denominator as a bunch of quotients. See below for examples.

To help you test, I have a program called **fraction_tester.cpp**, where you can enter commands on standard input to manipulate a fraction and test these methods. You call it with an optional prompt:

```
UNIX> bin/fraction_tester 'FT>'
FT> ?
QUIT           -- Quit the program.
?              -- Print commands.
CLEAR          -- Clear the fraction back to one.
CALCULATE      -- Calculate the fraction.
INVERT         -- Swap the numerator and denominator.
PRINT          -- Print the fraction as an equation.
MULTIPLY n     -- Multiply the fraction by n.
DIVIDE n       -- Divide the fraction by n.
MULT_FACT n    -- Multiply the fraction by n!
DIV_FACT n     -- Divide the fraction by n!
MULT_BINOM n k -- Multiply the fraction by n choose k.
DIV_BINOM n k  -- Divide the fraction by n choose k.
FT> PRINT
1
FT> MULTIPLY 5
FT> MULTIPLY 10
FT> PRINT
5 * 10
FT> CALCULATE
50
FT> DIVIDE 5    # Dividing by 5 removes 5 from the numerator.
FT> PRINT
10
FT> DIVIDE 5    # Dividing by 5 here adds 5 to the denominator.
FT> PRINT       # You only "cancel" numbers; you don't worry about factors
10 / 5
FT> MULT_FACT 6
FT> PRINT
2 * 3 * 4 * 6 * 10
FT> DIV_FACT 7
FT> PRINT
10 / 5 / 7
FT> CALCULATE
0.285714
FT> CLEAR
FT> MULT_BINOM 10 5
FT> PRINT
6 * 7 * 8 * 9 * 10 / 2 / 3 / 4 / 5
FT> CLEAR
FT> DIV_BINOM 10 5
FT> PRINT
2 * 3 * 4 * 5 / 6 / 7 / 8 / 9 / 10
FT> CALCULATE
0.00396825
```

```
FT> QUIT
UNIX>
```

When you compile your **src/fraction.cpp** with src/fraction_tester.cpp (which is done in the **makefile**), your output should match mine *exactly*. Many of the grading script's tests will use **fraction_tester** to test your program.

Your Job: #2: keno.cpp

Now that you've written **src/fraction.cpp**, use it to write **src/keno.cpp**, which calculates Keno odds using the formula above. **Keno.cpp** should read numbers on standard input using **cin**. The first number will be the amount to bet, and the second should be the number of balls that you are going to pick. After that, there will be pairs of numbers that compose the winning table. These will be in the order *catch, payout*, where **catch** is the exact number of balls to match, and **payout** is the payout of that number of balls. You do not need to error check.

So, for example, the "5 catch win" above would be represented with:

```
0.25 5    3 0.25   4 1   5 200
```

Now the first line of your program should print out the bet, and the second line should print the balls picked. The bet should be padded to two decimal places (use **printf** instead of **cout**). Then for each payout, you should print the probability of winning and the expected return (probability times payout). Use **cout** for these lines and have them match mine exactly. Finally, you should print the expected return per bet, which is the sum of all expected returns minus the bet. Have this padded to two decimal places (use **printf** instead of **cout**). Finally, print the normalized return, which is the expected return divided by the bet. Again, pad that to two decimal places.

This lets us evaluate all the Keno options in the flier. First, the "2-bit menu":

```
UNIX> echo "0.25 5    3 0.25   4 1   5 200" | bin/keno
Bet: 0.25
Balls Picked: 5
  Probability of catching 3 of 5: 0.0839351 -- Expected return: 0.0209838
  Probability of catching 4 of 5: 0.0120923 -- Expected return: 0.0120923
  Probability of catching 5 of 5: 0.000644925 -- Expected return: 0.128985
Your return per bet: -0.09
Normalized: -0.35
UNIX> echo "0.25 6   3 .25  4 .50   5 10   6 500" | bin/keno
Bet: 0.25
Balls Picked: 6
  Probability of catching 3 of 6: 0.12982 -- Expected return: 0.0324549
  Probability of catching 4 of 6: 0.0285379 -- Expected return: 0.014269
  Probability of catching 5 of 6: 0.00309564 -- Expected return: 0.0309564
  Probability of catching 6 of 6: 0.000128985 -- Expected return: 0.0644925
Your return per bet: -0.11
Normalized: -0.43
UNIX> echo "0.25 7   4 .25  5 .75   6 75   7 3000" | bin/keno
Bet: 0.25
Balls Picked: 7
  Probability of catching 4 of 7: 0.052191 -- Expected return: 0.0130477
  Probability of catching 5 of 7: 0.0086385 -- Expected return: 0.00647888
  Probability of catching 6 of 7: 0.000732077 -- Expected return: 0.0549058
  Probability of catching 7 of 7: 2.44026e-05 -- Expected return: 0.0732077
Your return per bet: -0.10
Normalized: -0.41
UNIX> echo "0.25 8   5 1   6 20   7 300 8 10000" | bin/keno
Bet: 0.25
Balls Picked: 8
  Probability of catching 5 of 8: 0.0183026 -- Expected return: 0.0183026
  Probability of catching 6 of 8: 0.00236671 -- Expected return: 0.0473343
  Probability of catching 7 of 8: 0.000160455 -- Expected return: 0.0481365
  Probability of catching 8 of 8: 4.34566e-06 -- Expected return: 0.0434566
Your return per bet: -0.09
Normalized: -0.37
UNIX> echo "0.25 9   5 1   6 10    7 35   8 800   9 13000" | bin/keno
Bet: 0.25
Balls Picked: 9
  Probability of catching 5 of 9: 0.0326015 -- Expected return: 0.0326015
  Probability of catching 6 of 9: 0.00571956 -- Expected return: 0.0571956
  Probability of catching 7 of 9: 0.000591678 -- Expected return: 0.0207087
  Probability of catching 8 of 9: 3.25925e-05 -- Expected return: 0.026074
  Probability of catching 9 of 9: 7.24277e-07 -- Expected return: 0.0094156
Your return per bet: -0.10
Normalized: -0.42
UNIX> echo "0.25 10  5 .25  6 2.50  7 25  8 250  9 2500  10 25000" | bin/keno
Bet: 0.25
Balls Picked: 10
  Probability of catching 5 of 10: 0.0514277 -- Expected return: 0.0128569
  Probability of catching 6 of 10: 0.0114794 -- Expected return: 0.0286985
  Probability of catching 7 of 10: 0.00161114 -- Expected return: 0.0402786
  Probability of catching 8 of 10: 0.000135419 -- Expected return: 0.0338548
  Probability of catching 9 of 10: 6.12065e-06 -- Expected return: 0.0153016
  Probability of catching 10 of 10: 1.12212e-07 -- Expected return: 0.0028053
Your return per bet: -0.12
Normalized: -0.46
UNIX>
```

Clearly, the "2-bit" menu is not a sound investment strategy. I especially love the part that says "Don't know what number to play? We have Quick Pick!" How about you just remove money from my bank account and then I don't have to think at all!!!!!!

How about the other Keno games? "You asked for it -- The Catch All 5 Spot":

```
UNIX> echo "1.50 5  5 1300" | bin/keno
Bet: 1.50
Balls Picked: 5
  Probability of catching 5 of 5: 0.000644925 -- Expected return: 0.838402
Your return per bet: -0.66
Normalized: -0.44
UNIX>
```

The "Brand New 8 spot:"

```
UNIX> echo ".40 8  5 2   6 20  7 200  8 20000" | bin/keno
Bet: 0.40
Balls Picked: 8
  Probability of catching 5 of 8: 0.0183026 -- Expected return: 0.0366052
  Probability of catching 6 of 8: 0.00236671 -- Expected return: 0.0473343
  Probability of catching 7 of 8: 0.000160455 -- Expected return: 0.032091
  Probability of catching 8 of 8: 4.34566e-06 -- Expected return: 0.0869132
Your return per bet: -0.20
Normalized: -0.49
UNIX>
```

And "100 Dimes" -- note this one has a catch zero:

```
UNIX> echo ".10 7  0 .10  6 20  7 1200" | bin/keno
Bet: 0.10
Balls Picked: 7
  Probability of catching 0 of 7: 0.121574 -- Expected return: 0.0121574
  Probability of catching 6 of 7: 0.000732077 -- Expected return: 0.0146415
  Probability of catching 7 of 7: 2.44026e-05 -- Expected return: 0.0292831
Your return per bet: -0.04
Normalized: -0.44
UNIX>
```

Perhaps I should have ridden the gondola........

This is no longer required -- I just include it for amusement

You are visiting your aunt and uncle, and as you walk by their computer room, you hear your eight-year-old neice Pippy swearing up a storm in front of her computer. As you probe, you realize that she's blowing a simulated wad of money on Free Online Keno!!!

Using your newfound understanding of Keno and your ability to program, write up a paragraph that shows Pippy how much she is losing on each of her fifteen potential Keno bets.

Doing Keno in Stages

Keno.cpp is a very small program (mine is 56 lines), but you can still do it in stages. For example, I did it in three stages. Stage 1 just prints out the input:

```
UNIX> echo "0.25 5    3 0.25   4 1   5 200" | bin/keno1
Bet: 0.25
Balls Picked: 5
  Catch 3/5 - 0.25
  Catch 4/5 - 1.00
  Catch 5/5 - 200.00
UNIX>
```

Stage 2 calculates the probabilities.

```
UNIX> echo "0.25 5    3 0.25   4 1   5 200" | bin/keno2
Bet: 0.25
Balls Picked: 5
  Probability of catching 3 of 5: 0.0839351
  Probability of catching 4 of 5: 0.0120923
  Probability of catching 5 of 5: 0.000644925
UNIX>
```

And then stage three finishes everything up.