CS 461
Spring 2022
Program 1

For this program you'll be doing a simple simulation to estimate your likelihood of winning in a game of poker.

Poker is a card game using a standard 52-card deck.[1] There are many variations, but as our focus here is on applying statistical learning rather than playing a game, we'll restrict ourselves to the simplest form —5 cards, nothing wild, no need to build the best hand possible out of 7 cards, etc.

Assume the game has 6 players – 'you' and 5 others. Your program will carry out the following actions:

- Repeatedly (500-1000 times):
    - Shuffle the 52-card deck, and deal yourself a 5-card hand.
    - Repeatedly (500-1000 times):
        - Using the remaining 47 cards, deal the other 5 players their hands
        - Determine if you would win or lose that hand; that is, if your hand would rank highest.[2] Update some counters accordingly.
        - Reshuffle the deck of 47 cards
    - Record the proportion of the above hands which you won.
- For each rank of hand, report the percentage of hands having that rank, and the average winning percentage (average of the averages) for each rank.

Your program should produce 2 output files:

- A session log output as a CSV (comma-separated value) file, with each hand on a separate line. For each hand: the cards in the hand; what the hand was evaluated as; and its winning percentage.
- A summary showing the percentage of hands falling into each rank, and the overall win percentage for each rank, as a 'normal' text file. Don't just list the percentages; add enough text to make it reader-friendly.

Your program may be written in your choice of: C++, C#, Java, Python.

You may submit your project as source code or zipped project folder, or via a link to GitHub, BitBucket, or other online repository.

Programming note (and digression, TL:DR version below):

The system random-number generator (random() and related functions) for the most common development environments will be more than adequate for this project. Early compilers had particularly weak RNGs, but all modern languages use the Mersenne Twister algorithm or a cryptographic primitive operation to generate random variates. These are statistically indistinguishable from truly random numbers.

Early compilers used *linear congruential operators*; equations of the form $x_{n+1} = [(x_n * r) + c] \bmod M$. It's not obvious, but carefully-chosen values for r, c, and M can produce a sequence of maximal length, where each integer from 0 to M-1 appears exactly once before any repeat. In effect, the operator defines a permutation of the integers from 0 to M-1, and the seed value just determines

1 See Appendix 1 for a description of the deck if you're not familiar with it.
2 See Appendix 2 for a listing of the values of poker hands.

where you start in that permutation. Needless to say, while compact and fast, this isn't particularly random, though it may be quasi-random *enough* for a given application.

You can find plenty of articles online decrying the RNGs in compilers, particularly Visual Studio. Those are mostly out of date, or are talking about the system RNG in a security context; no purely-software RNG is adequate for a security application. The issue is that if an attacker can obtain a long series of the RNG's output, it is possible to start making deductions about the internal state of the generator, thus enabling at least partial prediction of future output—and those outputs are often used as session keys. Thus, some source of entropy, of true randomness, must be injected into the sequence. Every x86 processor since the Pentium III has had a hardware RNG, accessed via the URAND() function. The thermometer on the processor is accurate to about a tenth of a degree, but the temperature registers go out to about a hundred-thousandth of a degree; the least-significant digit is taken as random, as it's determined by quantum fluctuations on the chip, tiny changes in temperature, and air turbulence inside the computer's chassis. With our current state of knowledge, those are unpredictable enough to be considered 'truly' random. There are a few other thoughts on "how random is random?" and "how random is random enough?" on this page.

TL:DR version: Use the system's random number generator. I've had students tie themselves into knots trying to get an overly-complex RNG working when it wasn't necessary; or, worse, read a website that says Visual Studio's RNG is no good, so they're using an implementation of Mersenne Twister they found online...when VS uses the same algorithm.

APPENDIX 1: The standard deck.

The standard (US) deck of playing cards contains 4 suits—Spades, Hearts, Diamonds, Clubs—each of which contains 13 cards, ranging in value from the Ace (1) to 10, plus the *court cards* (or 'face cards') of the Jack, Queen, and King. In Poker, the Ace may count as the lowest value (1) or the highest, above the King. Cards are usually indicated by a short string consisting of the rank then the suit, with the rank being indicated by one of: A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, and the suit by S, H, D, C. Thus 6D denotes the 6 of Diamonds, KS the King of Spades, etc.

APPENDIX 2: The value of the hands.

Poker hands consist of 5 cards, dealt from a well-shuffled deck. These are combinations, not permutations; the order the cards are dealt is not significant. The usual rules allow ties to be broken readily; if two hands are truly tied, count it as half a win. They're rare, but they do happen.

The rank of the hands, from lowest to highest:

- No pair (high card): The hand does not qualify for any of the other ranks listed. Ties are broken by looking at the highest-ranking card in the hand. If the highest cards are tied, look at the second-highest, and so on.
- One pair: Two cards are of the same rank; the other 3 cards have different ranks. Ties broken by comparing values of the pair. If two hands have one pair at the same rank, compare the highest of the remaining 3 cards, as for no pair.
- Two pair: Two cards with the same rank; two cards with the same rank different from the first pair; and an unrelated card. Break ties by comparing the higher pair; then the lower pair; then the unrelated card.
- Three of a kind: Three cards of the same rank, and two unrelated cards. Ties broken by comparing the values of the 3 of a kind. (It's not possible for 2 hands to each have 3 cards of the same rank.)
- Straight: 5 cards forming an unbroken sequence; not all cards are the same suit. The Ace can count as high (A-K-Q-J-10) or low (5-4-3-2-A). It is *not* possible to go 'around the corner,' i.e. 3-2-A-K-Q is NOT a straight. Ties broken by comparing highest card in sequence.

- Flush: All 5 cards are the same suit and they do not form a sequence. Ties broken by comparing card ranks, as for no-pair hands; there is no relative ranking of the suits themselves.
- Full House: Three of a kind plus a pair: three cards of one rank, and 2 cards of another rank. Ties broken by comparing the values of the three cards.
- Four of a Kind: 4 cards one one rank, and 1 unrelated card. Ties broken by comparing values of the 4 of a kind.
- Straight Flush: A straight, and a flush: 5 cards in an unbroken sequence, all the same suit. A Straight Flush headed by the Ace as high card (A-K-Q-J-10) is a *royal flush*. The likelihood of a royal flush in 5 cards is about 1 in 650,000 (the exact odds are 649,739:1). In a best-5 game like Texas Hold ‘Em, the odds are ‘only’ about 39,000 to 1 against.