CS 241 Data Organization using C

Project:
Identifying the Rank of a Poker Hand and an Empirical Calculation of Probabilities

Instructor: Joel Castellanos
  e-mail: joel@unm.edu
  Web: http://cs.unm.edu/~joel/

Empirical

Based on, concerned with, or verifiable by observation or experience rather than theory or pure logic.

Consider the Probability of drawing an ace from a well shuffled, standard 52-card deck of playing cards.

A pure logic approach reasons: There are 4 aces in the 52 card deck. Therefore, there is a 4 in 52 chance of drawing an ace = 4/52 = 2/26 = 1/13 or a probability of about 0.0769.

In an empirical approach, one might shuffle a deck some large number of times (say 1 million). After each shuffle, draw one card, and record whether it is an ace. Then, we expect the future probability of drawing an ace to be approximately equal to the observed ratio of aces draw to the total number of draws.
Empirical methods of gaining scientific knowledge require making a large number of observations.

John rolls a 6-sided die. The die comes to rest with the three facing up. Therefore, concludes John, when a 6-sided die is rolled, the result is 3.

British author, Kazuo Ishiguro, was born in Japan. He emigrated to England in 1958 at the age of 4. In a 2015 interview, he commented: "None who I encountered in my school days had ever before met, read books or watched movies of someone of Japanese descent. They had no prejudice. I quickly learned that the impression made in the first few minutes of each new encounter would persist for a long time."

Not all Statements are Created Equal

A 1999 commencement speech at Andrews University in Michigan was delivered by an M.D. from the University of Michigan Medical School who was at that time serving as Director of Pediatric Neurosurgery at Johns Hopkins Hospital.

"....My own personal theory is that Joseph [an important figure in the Bible's Book of Genesis] built the pyramids to store grain...."

A logical, but uninformed person, seeing a pyramid for the first time, might, for a short while, postulate such a theory.

It is illogical to continue holding that theory after it has been pointed out that the interior space is very small.
**Five Card Draw: Project Overview**

**Input:**
1) A text file where each line represents a 5 card poker hand.
2) Each line is independent of other lines.

**Output:**
Echo each line of input, followed, on the same line, by:
1) The *rank* of the input hand as a text label such as "Full House" or "3 of a Kind".
2) For each card in the hand, the probability of getting an improved hand by discarding that one card and drawing a replacement from the cards remaining in a standard, 52 card, poker deck.

---

**Input Format**
- A text file where each line is processed independently and represents a poker hand.
- Each correctly formatted line is a space delimited set of five pairs of characters.
- The first character in each pair represents the card rank and must be '0' (representing 10) or '2' through '9', (representing a card with the given number), 'A' (ace), 'J' (jack), 'Q' (queen), or 'K' (king).
- The second character in each pair represents the card suit and must be 'C', 'D', 'S' or 'H'. Each character representing Clubs, Diamonds, Spades or Hearts, respectively.
Standard 52-card deck

The Standard 52-card deck of playing cards includes thirteen ranks of each of the four suits: clubs (♣), diamonds (♦), hearts (♥) and spades (♠).

Errors

Print the label “Error” for each input line that does not make a valid hand. Possible errors are:

- A line that does not contain exactly five pairs of characters separated by a single space.
- A line that contains a pair of characters that does not correspond to a card as defined in the specifications.
- A line that does not end with the newline character, ‘/n’.
- A line that contains two or more of the same pairs of characters.
Sample Input and Output: Error Examples

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC 3S 5S 1H 9D</td>
<td>AC 3S 5S 1H 9D &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>AC 3S 5S 9D</td>
<td>AC 3S 5S 9D &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>5C 3S 5S 4S 5D</td>
<td>5C 3S 5S 4S 5D &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>5C 3S 5 4S 5D</td>
<td>5C 3S 5 4S 5D &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>5C 3S 5S 4S 5D</td>
<td>5C 3S 5S 4S 5D &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>5C 3S 5S 4S 3S</td>
<td>5C 3S 5S 4S 3S &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>5C 3S 5S 4S 2S 2D</td>
<td>5C 3S 5S 4S 2S 2D &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>5C 3S 10S 4S 2S</td>
<td>5C 3S 10S 4S 2S &gt;&gt;&gt;Error</td>
</tr>
</tbody>
</table>

Card Rank in Five Card Draw Poker

Cards in poker are ranked, from highest to lowest:

A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3 and 2.

However, when an ace is used to form a straight or straight flush from ace-to-five, then the ace is lowest ranked card (an ace-to-five straight is called a baby straight).

Suits do not have rank: the ace of clubs is equal in rank to the ace of spades.
Poker Rank Names (in rank order)

- **Straight Flush**: Five cards in sequence, of the same suit.
- **Four of a Kind**: Four cards of the same rank.
- **Full House**: Three cards of the same rank, and two cards of a different, matching rank.
- **Flush**: Five cards of the same suit.
- **Straight**: Five cards in rank sequence.
- **Three of a Kind**: Three cards of the same rank.
- **Two Pair**: Two sets of two cards of the same rank.
- **One Pair**: Two cards of the same rank.
- **High Card**: All cards different rank and more than one suit

**Error**: The input line contains an error.

If Same *Type Rank*, Higher *Card Rank* Wins

- If two hands have the same classification, the higher rank *in the classification* wins:
  - Straight: $5\spadesuit, 6\spadesuit, 7\spadesuit, 8\spadesuit, 9\spadesuit$ beats $3\heartsuit, 4\heartsuit, 5\heartsuit, 6\heartsuit, 7\heartsuit$
  - 3-of-a-Kind: $J\spadesuit, J\spadesuit, J\spadesuit, 2\spadesuit, 3\spadesuit$ beats $7\spadesuit, 7\spadesuit, 7\spadesuit, A\spadesuit, K\spadesuit$
  - High Card: $J\spadesuit, 2\spadesuit, 3\spadesuit, 4\spadesuit, 5\spadesuit$ beats $10\spadesuit, 9\spadesuit, 8\spadesuit, 7\spadesuit, 5\spadesuit$

- Cards not part of the classification are ignored:
  - 2-of-a-Kind: $J\spadesuit, J\spadesuit, A\spadesuit, K\spadesuit, 3\spadesuit$ ties $J\heartsuit, J\heartsuit, 7\spadesuit, 5\spadesuit, 6\spadesuit$
  - 2-pair: $J\spadesuit, J\spadesuit, 5\spadesuit, 5\spadesuit$ $2\spadesuit$ ties $J\heartsuit, J\heartsuit, 5\spadesuit, 5\spadesuit, A\spadesuit$

- If two-pair, the highest pair breaks ties:
  - 2-pair: $J\spadesuit, J\spadesuit, 2\spadesuit, 2\spadesuit, 5\spadesuit$ beats $9\heartsuit, 9\spadesuit, 8\spadesuit, 8\spadesuit, K\spadesuit$
  - 2-pair: $J\spadesuit, J\spadesuit, 5\spadesuit, 5\spadesuit, 2\spadesuit$ beats $J\heartsuit, J\heartsuit, 2\spadesuit, 2\spadesuit, 6\spadesuit$
Sample Input and Output

<table>
<thead>
<tr>
<th>in:</th>
<th>out:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D 2C 5H 2H 2S</td>
<td>2D 2C 5H 2H 2S &gt;&gt;&gt;Four of a Kind 0.0% 0.0% 0.0% 0.0% 0.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in:</th>
<th>out:</th>
</tr>
</thead>
<tbody>
<tr>
<td>6C 5C 8D 7C 4S</td>
<td>6C 5C 8D 7C 4S &gt;&gt;&gt;Straight 0.0% 0.0% 0.0% 0.0% 8.5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in:</th>
<th>out:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7C 6C 8D 5C 4C</td>
<td>7C 6C 8D 5C 4C &gt;&gt;&gt;Straight 0.0% 0.0% 19.1% 0.0% 8.5%</td>
</tr>
</tbody>
</table>

Suits are Equal

- If two hands have the same classification and rank, then the hands are of equal value:
  - Straight Flush: 5♠, 6♠, 7♠, 8♠, 9♠
  - ties: 5♦, 6♦, 7♦, 8♦, 9♦
Special Cases:

The Ace can rank as the lowest or highest card in a straight:
{A♦, 2♦, 3♦, 4♦, 5♣} or {10♣, J♣, Q♣, K♣, A♦}

An ace cannot be in the middle of a straight. For example,
{J♣, Q♣, K♣, A♦, 2♣} is not a straight.

Using Logic to Check Empirical Results

Give the Hand {5♣, 5♦, 5♥, 9♦, 3♠}

This hand is 4 of a kind. There are two ways it could be improved with a single card swap: 4-of-a-kind (by replacing the 9 or 3 with the 5♠) or Full-house (replacing the 9 with a 3 or the 3 with a 9).

After dealing 5 cards, the deck has 47 cards remaining.

Total probability of improving when discarding any 5 is zero and the probability of improving when discarding the 9 or the 3, equals \((1 + 3)/47 = 0.0851063829787234 = \text{about, 8.5%} \). Therefore, after 1 million times of replacing the 9, the hand should be improved about \(0.0851 \times 1,000,000 = 85,000\) times.
Given the Hand \{5♣, 7♦, 8♦, 9♦, 10♦\}

If you discard the 5♣, you get an improved hand if you draw:

- Straight (8 cards): 6♦ 6♥ 6♣ 6♠ J♦ J♥ J♣ J♠
- Flush: (7 cards): 2♦ 3♦ 4♦ 5♠ 6♠ 7♠ 8♠
- Pair: (12 cards): 7♥ 7♠ 7♣ 8♥ 8♠ 8♦ 9♥ 9♠ 9♦ 10♥ 10♠ 10♣
- High Card: (9 cards): J♦ J♥ J♣ J♠ Q♦ Q♥ Q♣ Q♠ K♦ K♥ K♣ A♦ A♥ A♠

Total cards that will improve = 8+7+12+9  
= 36 in 47  = 76.6%

Useful Global Fields

Near top of program, outside all functions:

```c
#define DECK_SIZE 52
#define HAND_SIZE 5
#define SUIT_COUNT 4

const char SUIT_LIST[] = "CDHS";
const int NO_CARD = -1;

int handRank[HAND_SIZE];
char handSuit[HAND_SIZE];
```

Parallel arrays:
- `handRank[i]`
- `handSuit[i]`

Reference the same card.
Helper Functions (1 of 3)

```c
int readLine(void)
{  //Read character-by-character until \n or EOF.
    //Sets handRank and handSuit.
    //Return 0 if syntax error.
    //Return EOF if EOF.
    //Return 1 if good.
}

int trashToEndOfLine(void)
{  //Called by readLine when error is found.
}
```

Other Helper Functions (2 of 3)

```c
void sortCards(void)
{  //Sort by rank, and for same rank, sort by suit.
    //Thus, identical hands will always have the same order.
    //Sorted Example: A♦, K♣, 2♠, 2♦, 2♣

    //There are 311,875,200 ways to deal five cards from the
    // deck but only 2,598,960 distinct hands, because the
    // order in which cards are dealt or arranged does not
    // matter.
}
```
Other Helper Functions (3 of 3)

```c
void printHand(void)
int isFlush(void)
int isStraight(void)
int is4OfAKind(void)
```
stdlib.h: The rand Function

#include <stdlib.h>

int rand(void)
Generate a uniformly distributed pseudo-random value between 0 and RAND_MAX.

On moons.cs.unm.edu: RAND_MAX = 2,147,483,647
On many older machines: RAND_MAX = 32,767

void srand (unsigned long seed)
Initializes pseudo-random number generator.

If no seed value is provided, the rand() function is automatically seeded with a value of 1.

Usually, called once and only once in a program.

rand() to get an integer [0, n-1]

#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    srand((unsigned long)time(NULL));
    int i;
    for (i=0; i<20; i++)
    {
        int r = rand(); // [0,RAND_MAX]
        int roll = r%6; // [0,5]
        printf("%d (%d)\n", roll, r);
    }
}
What are the Properties of the Output?

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int i;
    int bin[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    srand((unsigned long)time(NULL));

    for (i=0; i<1000000; i++)
    {
        int r = (rand()%6) + (rand()%6);
        bin[r]++;
    }

    for (i=0; i<=10; i++)
    {
        printf("bin[%2d] = %7d\n", i, bin[i]);
    }
}
```
rand() to get a double \([0.0, 1.0]\)

```c
#include <stdio.h>
#include <stdlib.h>

double randomDouble()
{
    return (double)rand() / (double)RAND_MAX;
}

void main(void)
{
    srand((unsigned long)time(NULL));
    int i;
    for (i=0; i<20; i++)
    {
        printf("%f\n", randomDouble());
    }
}
```

A Path To Success

- Solve a hard problem by breaking it down into smaller problems.
- Solve each smaller problem – one at a time.
- Decide what output to produce at the end of each sub-problem to prove to yourself that you actually solved the sub-problem. *Then, produce that output.*
- If you fail to produce what you expect, then add print statements throughout the sub-process to figure out what is actually happening.
Echo Input & Find Bad Data

To get started, your first version of this program should do nothing more than read the given input file and for each line of input, echo the input line followed by either " >>>Error" or "good".

Only the second version should worry about finding the error of having a repeated card (which is a semantic error, not a syntax error).

Next, get your program to correctly identify "two of a kind".

Each following version identifying just one more rank.

Only after you have correctly identified all the hands, should you start worrying about the probabilities!

Write Testing Functions:

- Deal 1 million random hands.
  - Count the number of each rank. Verify that the numbers are approximately equal.
  - Count the number of each suit. Verify that the numbers are approximately equal.
  - Verify that no hand contains more than one of the same card.

- Try using a different seed and generate another 1 million cards. Verify that the counts are different.
Project Grading Rubric

4 Points:  Your program is named poker_yourFirstName_yourLastName.c and correctly attached in blackboardLearn. Of course, substitute your actual name.

72 Points: Two points for each passed test of: poker.in. One point for each correctly ranked hand and 2 points for an output line that matches each character in the corresponding line of poker.out. A "match" for the percentages is ±1 in the tenths place.

24 Points: Two points for each passed test of: pokerUnknown.in, which, you do not get to see until after your assignment is graded.

Up to -20 Points: Source code does not follow the CS-241 coding and comment standard.

-80 Points: Program does not use the required empirical method.

Method Improvement for Future

What algorithm would give the same results AND be:

- Easier to code,
- Run vastly faster, and
- More accurate?