CS 241 Data Organization using C

Project 1: Naming a Poker Hand and Predicting some outcomes.

Instructor: Joel Castellanos
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/
Office: Farris Engineering Center (FEC) room 319

Project 1: Overview

Input:
- A text file where each line is a list of unique numbers: 1 through 52.
- Each number represents a particular card in a standard deck of playing cards.

Output:
Echo each line of input, followed, on the same line, by:

1) The **hand label** that best names the set of input.
2) For each input card, the probability of getting an improved hand by discarding that card and drawing a replacement.
Project 1: Hand Labels

Error: The input line contains an error.

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.

---

Project 1: Card Numberings

The numbers 1 through 52 represent each of 13 ranks (Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King) in each of 4 suits (Clubs, Diamonds, Hearts, Spades):

<table>
<thead>
<tr>
<th>Card Number</th>
<th>Rank</th>
<th>Suit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ace</td>
<td>♣Clubs</td>
</tr>
<tr>
<td>2</td>
<td>Ace</td>
<td>♦Diamonds</td>
</tr>
<tr>
<td>3</td>
<td>Ace</td>
<td>♥Hearts</td>
</tr>
<tr>
<td>4</td>
<td>Ace</td>
<td>♠Spades</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>♣Clubs</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>♦Diamonds</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>♥Hearts</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>♠Spades</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>King</td>
<td>♠Spades</td>
</tr>
</tbody>
</table>

In rank:

- Ace = 1 or 14
- Jack = 11
- Queen = 12
- King = 13
### Sample Input and Output: Error Examples

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>26,30,34,28,0</td>
<td>26,30,34,28,0 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>45,46,47,48,49</td>
<td>45,46,47,48,49 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>45,46,47,48</td>
<td>45,46,47,48 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>24,12,20,16</td>
<td>24,12,20,16 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>20,19,17,53,18</td>
<td>20,19,17,53,18 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>17,20,18,17,16</td>
<td>17,20,18,17,16 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>2,19,17,20,2</td>
<td>2,19,17,20,2 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>17,20,13,16</td>
<td>17,20,13,16 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>17,20,18,13,16</td>
<td>17,20,18,13,16 &gt;&gt;&gt;Error</td>
</tr>
<tr>
<td>17,20,18,13,16,11</td>
<td>17,20,18,13,16,11 &gt;&gt;&gt;Error</td>
</tr>
</tbody>
</table>

### Sample Input and Output

#### in: 13, 5,14, 15, 16
#### out: 13, 5,14, 15, 16 >>>Four of a Kind 0.0% 0.0% 0.0% 0.0% 0.0%

#### in: 25, 21,30,17,16
#### out: 25, 21,30,17,16 >>>Straight 0.0% 0.0% 0.0% 0.0% 8.5%

#### in: 25, 21, 30, 17, 13
#### out: 25, 21, 30, 17, 13 >>>Straight 0.0% 0.0% 19.1% 0.0% 8.5%
If Same Classification, Higher Rank Wins

- If two hands have the same classification, the higher rank in the classification wins:
  
  Straight: 5♣, 6♣, 7♣, 8♣, 9♦ beats 3♥, 4♥, 5♦, 6♦, 7♠
  
  3-of-a-Kind: J♣, J♦, J♠, 2♣, 3♥ beats 7♣, 7♦, 7♠, A♣, K♥
  
  High Card: J♣, 2♦, 3♣, 4♠, 5♥ beats 10♠, 9♦, 8♠, 7♣, 5♦

- Cards not part of the classification are ignored:
  
  2-of-a-Kind: J♣, J♦, A♠, K♣, 3♥ ties J♥, J♦, 7♣, 5♣, 6♦
  
  2-pair: J♣, J♦, 5♣, 5♠, 2♥ ties J♥, J♦, 5♥, 5♥, A♠

- If two-pair, the highest pair breaks ties:
  
  2-pair: J♣, J♦, 2♣, 2♥, 5♥ beats 9♥, 9♦, 8♠, 8♣, K♣
  
  2-pair: J♣, J♦, 5♣, 5♠, 2♥ beats J♥, J♦, 2♣, 2♥, 6♣

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.

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 numbers separated by commas. There may or may not be any number of spaces before or after each number.
- A line that contains a number less than 1 or greater than 52.
- A line that contains any character other than a digit, space or a comma.
- A line that does not end with the newline character, ‘/n’.
- A line that contains two or more of the same number.
**Probability Example: High Card**

Give 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♦ 9♦ K♦ A♦
- 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♣ K♠ A♦ 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 handNum[HAND_SIZE];
int handRank[HAND_SIZE];
char handSuit[HAND_SIZE];
```

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

All reference the same card.

#define can be used to declare array sizes. const int cannot.
Helper Function: numToRank

```c
int numToRank(int num)
{
    if (num < 1 || num > DECK_SIZE) return -1;
    return 1 + (num-1)/SUIT_COUNT;
}
```

Helper Function: numToSuit

```c
char numToSuit(int num)
{
    if (num < 1 || num > DECK_SIZE) return -1;
    int suitIdx = (num-1) % SUIT_COUNT;
    return SUIT_LIST[suitIdx];
}
```
Other Helper Functions (1 of 2)

```c
int rankSuitToNum(int rank, char suit)

int readLine(void)
{    //Read character-by-character until \n or EOF.
    //Sets handNum, 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 2)

```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♣
}

int getNextCard(int currentCard)
{    //Return the smallest integer greater than currentCard
    // that is not in handNum.
}

void printHand(void)
int isFlush(void)
int isStraight(void)
int is4OfAKind(void)
```
Testing Helper Functions

```c
void testCardConversion(void)
{
    int i;
    for (i=1; i<=52; i++)
    {
        int rank = numToRank(i);
        char suit = numToSuit(i);
        int num   = rankSuitToNum(rank, suit);

        printf("%d=%d: %d%c\n",i,num, rank, suit);

        if (num != i)
        {
            printf("****ERROR****\n");
            exit(0);  //requires #include <stdlib.h>
        }
    }
}
```

Write Testing Functions:

- Generate 1 million random hands.
  - Convert the card numbers to rank and suit.
  - 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.
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

Create a method `readLine(void)` that reads and echoes one line of input and checks for syntax errors.

Work to find all errors except repeated cards, which are more semantic than syntax errors and are best checked later.

1) 13, 5,14, 15, 50 >>>OK
2) 20, 19,17, 44, 2 >>>OK
3) 26,30,34,28,40 >>>OK
4) 45,46,47,48 49 >>>Error  //Missing comma
5) 45,46,47,48 >>>Error     //Only 4 cards
6) 24, 12,20,16,28 >>>OK
7) 20, 19,17, 53, 18 >>>Error // Bad card number: 53
8) 17,20 ,18,13,16 >>>OK
9) 2, 19,17, 20, 2 >>>OK    // Didn’t find this semantic error
10) 17,20 x 18,13,16 >>>Error  // Bad character x
11) 17,20 , 18,13,16, >>>Error // Extra comma at end
12) 17,20,1 8,13,16 >>>Error  // Space between 1 & 8
Sort cardNum[

Create a method `sortCards(void)` to reorder global array, `int cardNum[MAX_CARDS]` sorted by card number.

This is a perfect place to check for repeated cards.

Note: When sorting 5 items, *easy to code* is better than fast.

1) 13, 5, 14, 15, 50 >>>5 13 14 15 50
2) 20, 19, 17, 44, 2 >>>2 17 19 20 44
3) 26, 30, 34, 28, 40 >>>26 28 30 34 40
4) 45, 46, 47, 48, 49 >>>Error
5) 45, 46, 47, 48 >>>Error
6) 24, 12, 20, 16, 28 >>>12 16 20 24 28
7) 20, 19, 17, 53, 18 >>>Error
8) 17, 20, 18, 13, 16 >>>13 16 17 18 20
9) 2, 19, 17, 20, 2 >>>Error // find any repeated cards.
10) 17, 20 x 18, 13, 16 >>>Error
11) 17, 20, 18, 13, 16, >>>Error
12) 17, 20, 18, 13, 16 >>>Error

---

Find Rank and Suit

Create two global arrays:

`int cardRank[MAX_CARDS];`
`char cardSuit[MAX_CARDS];`

Create a method `setRankAndSuit(void)`.

1) 13, 5, 14, 15, 50 >>>2C 4C 4D 4H 13D
2) 20, 19, 17, 44, 2 >>>1D 5C 5H 5S 11S
3) 26, 30, 34, 28, 40 >>>7D 7S 8D 9D 10S
4) 45, 46, 47, 48, 49 >>>Error
5) 45, 46, 47, 48 >>>Error
6) 24, 12, 20, 16, 28 >>>3S 4S 5S 6S 7S
7) 20, 19, 17, 53, 18 >>>Error
8) 17, 20, 18, 13, 16 >>>4C 4S 5C 5D 5S
9) 2, 19, 17, 20, 2 >>>Error
10) 17, 20 x 18, 13, 16 >>>Error
11) 17, 20, 18, 13, 16, >>>Error
12) 17, 20, 18, 13, 16 >>>Error
4 Points: Your program is named poker_yourFirstName_yourLastName.c and correctly attached in blackboardLearn. Of course, substitute your actual name.

66 Points: Two points each for each passed test of: poker.txt. A passed test is each line of output produced that matches each character in the corresponding line of pokerCorrect.out. Both of these files can be found on the class website.

30 Points: Two points each for each passed test of: pokerUnknown.txt, 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.