ECE 71 – Introduction to Computational Programming

Professor Kriehn – Fall 2017

Code Due By: Midnight on Tuesday, October 31, 2017
Write-up Due By: Class on Wed/Thu, November 1/2, 2017

HOMEWORK #25 – 2-D Arrays

Write a program that creates (and stores) a 2-D array of 64 random integers (8 rows and 8 columns) between 1 and 100. The program should then print out the elements of the array, as well as the element location (row and column) of the largest value(s) in the array.

Next rotate the array so that the rows become the columns and the columns become rows and store the rotated values in a second 2-D array. Print the rotated array, along with the location of the maximum value(s).

Specifications:
Use constants to define the number of rows and columns in your array (ROWS and COLS). Use the srand() function to seed the random number generator after reading in a seed value. The rand() function should be called multiple times to generate the integers – fill in the elements of the array row by row. Print the results with a width of 5.

Use the following function prototypes to help you program your solution:

```
void seedRNG(unsigned int seed);
// Seeds the Random Number Generator with the "seed" value

int randNum(void);
// Returns a random number between 1-100

void populateArray(int a[ROWS][COLS]);
// Populates an array with random values between 1-100

int findMax(int a[ROWS][COLS]);
// Returns the maximum value of the array

void printArray(int a[ROWS][COLS]);
// Prints the 2-D array

void printMax(int a[ROWS][COLS], int max);
// Prints the row, column location(s) of the maximum value(s)

void rotateArray(int a[ROWS][COLS], int b[ROWS][COLS]);
// Rotates array a[][] into b[][]
// ROW positions in a[][] become column positions in b[][]
// COLUMN positions in a[][] become row positions in b[][]
```

If you execute the program, the following information should be displayed:

```
~> main.o
Enter a random seed: 34

   59   69   75   67   39   90    7   81
   68   75   56   99   55   45   10   59
   56   79   62   45   28   10    8   16
   91   24   21   64   42   39   97  100
    7   23   19   45   64   25   78   31
   51   33   29    5   77   90   63   85
   20   76   81  100   85   89   67   27
   64   87   42    5   25   38   57   83

Maximum Value at Element: 3,7
Maximum Value at Element: 6,3

   59   68   56   91    7   51   20   64
   69   75   79   24   23   33   76   87
   75   56   62   21   19   29   81   42
   67   99   45   64   45    5  100    5
   39   55   28   42   64   77   85   25
   90   45   10   39   25   90   89   38
    7   10    8   97   78   63   67   57
   81   59   16  100   31   85   27   83

Maximum Value at Element: 3,6
Maximum Value at Element: 7,3
```

HOMEWORK #26 – 5 Card Stud Analyzer

Write a program that analyzes a poker hand for 5 card stud. The following link may be helpful if you are a bit unfamiliar with poker rules:

http://en.wikipedia.org/wiki/List_of_poker_hands

For this program, assume there are no jokers, and the highest hand possible is a "Royal Flush", which is a Straight Flush with an Ace High. Hands in order of decreasing value are thus:

Royal Flush
Straight Flush
Four of a Kind
Full House
Flush
Straight
Three of a Kind
Two Pairs
One Pair
High Card

Specifications:
Split your program into three functions:

```
void readCards(void);
// Read in 5 cards, Ignoring Bad/Repeated Cards

void analyzeHand(void);
// Analyze cards for Different Hands to Play

void printResults(void);
// Print the Best Hand
```

Because all three functions are void functions, we will use a number of global constants and variables:

Global Constants

```
NUM_RANKS                 // Stores number of ranks in a deck (13)
NUM_SUITS                 // Stores number of suits in a deck (4)
NUM_CARDS                 // Stores number of cards to be played (5)
```

Global Variables

```
int numInRank[NUM_RANKS]  // Array representing the number of a given rank in hand
                          // numInRank[0] represents number of 2's
                          // numInRank[1] represents number of 3's
                          // …
                          // numInRank[12] represents number of Aces

int numInSuit[NUM_SUITS]  // Array representing the number of a given suit in hand
                          // numInSuit[0] represents number of Clubs
                          // numInSuit[1] represents number of Diamonds
                          // numInSuit[2] represents number of Hearts
                          // numInSuit[3] represents number of Spades

bool Royal                // Set true if player has a Royal Flush
bool Flush                // Set true if player has a Flush
bool Straight             // Set true if player has a Straight
bool Four                 // Set true if player has Four of a Kind
bool Three                // Set true if player has Three of a Kind
int Pairs                 // Counts number of Pairs in Hand
```

Finally, a 2D Boolean array called cardExists[NUM_RANKS][NUM_SUITS] will be needed to be used in the readCards() function to keep track of which cards have been successfully entered into the program. The row represents the rank of the card (2 through Ace), and the column the suit (Clubs, Diamonds, Hearts, and Spades – in that order!). A given element in the 2D array should be set to true whenever a card is successfully read into the program. Assume the input is a number of letter representing the rank (2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A), followed by a letter for the suit (C, D, H, S). Letters may be input lowercase or uppercase.

The appropriate element in the numInRank[] array should be incremented every time a given rank is

input into the program. Force instance, if the 3 of Hearts and the 3 of Diamonds are input, numInRank[1] will be 2, since there are 2 3's in your hand. Likewise, the appropriate element in the numInSuit[] array should be incremented every time a given suit is input into the program. In the previous example, numInSuit[1] = 1 and numInSuit[2] = 1, since the order of the suits are Clubs, Diamonds, Hearts, and Spades. As cards are successfully read in, each of these arrays should be adjusted, as appropriate.

Once the cards are read in, the numInRank[] and numInSuit[] arrays can be used to determine your possible hands (i.e. if you have a straight, a flush, 3 of a Kind, etc.) within the analyzeHand() function. The appropriate global variables should be set as appropriate.

Finally, the printResults function can analyze these variables to determine your best hand. If you have a straight and a flush, but not Ace high, then your best hand is a Straight Flush.

If you execute the program, the following information should be displayed:

```
~> main.o
Welcome to the 5 Card Stud Analyzer Program

Enter Card 1: 3hasdfa
Ignoring bad card.

Enter Card 1: h3
Ignoring bad card.

Enter Card 1: 3h
Enter Card 2: 4H
Enter Card 3: 4d
Enter Card 4: 7h
Enter Card 5: 8C

Best Hand: One Pair
~> main.o
Welcome to the 5 Card Stud Analyzer Program

Enter Card 1: KC
Enter Card 1: QC
Enter Card 1: QC
Ignoring duplicate card.

Enter Card 3: JC
Enter Card 4: AC
Enter Card 5: TC

Best Hand: Royal Flush
```

Develop your I/O diagram and pseudocode, debug your code, and submit to the Grader Program.