Program Set #1
Total Points: 30

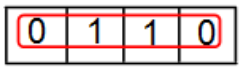
Three problems must be implemented for full credit. The required (starred) problem marked in the set must be implemented by everyone. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points each)

Note: If coding in C++, use the STL string class for problems involving strings. Do not use C style strings.

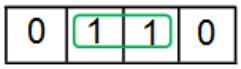
Section One- Choose two problems.

1. Write a program that finds the minimal number of groupings in a matrix. The user will enter the size of the rows and columns in the range `[2,8]` from the keyboard. Error check input size. The program then will randomly fill the numbers 0 and 1 into the matrix and outputs to the screen the matrix and the minimum number of rectangles/squares that cover all the 1 values in the matrix. The size of a rectangle/square is defined by the number of 1's in it. Here are the rules in forming groups of ones:

- Each group should contain the largest number of 'ones' and no blank cells.



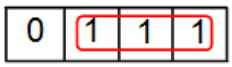
Incorrect



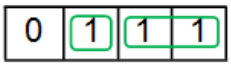
Correct

- The number of 'ones' in a group must be a power of 2 i.e. a group can contain:

$16(=2^4)\ or\ 8(=2^3)\ or\ 4(=2^2)\ or\ 2(=2^1)\ or\ 1(=2^0)\ cells$

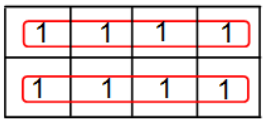


Incorrect

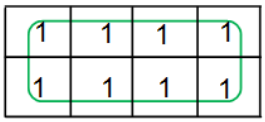


Correct

- Grouping is carried-on in decreasing order meaning, one must try to group for 8 (octet) first, then for 4 (quad), followed by 2 and lastly for 1 (isolated 'ones').

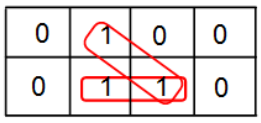


Incorrect

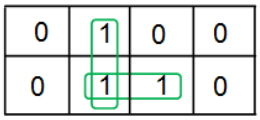


Correct

- Grouping is done either horizontally or vertically or in terms of rectangles/squares. Diagonal grouping of 'ones' is not permitted.

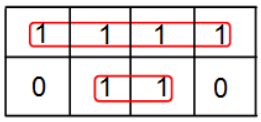


Incorrect

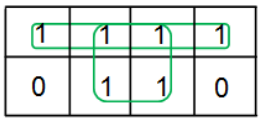


Correct

- The same element(s) may repeat in multiple groups only if this increases the size of the group.

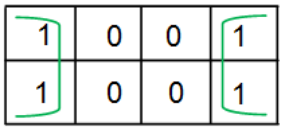


Incorrect



Correct

- The elements around the edges of the matrix, including the four corners, are adjacent and can be grouped together.



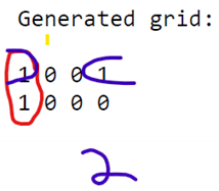
In review:

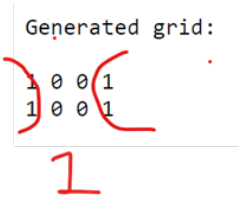
- No zeros allowed.
- No diagonals.
- Only power of 2 number of cells in each group.
- Groups should be as large as possible.
- Every 1 must be in at least one group.
- Overlapping and wrap around allowed.
- Get the fewest number of groups possible.

Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

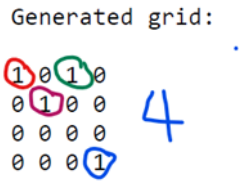
Sample Run:

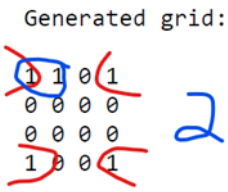
```
Enter the number of rows (2-10): 2
Enter the number of cols (2-10): 4

 Generated grid:

 1 0 0 1
 1 0 0 0

 The least number of rectangles/squares formed is 2

Run Again (Y/N): y

Enter the number of rows (2-10): 2
Enter the number of cols (2-10): 4

 Generated grid:

 1 0 0 1
 1 0 0 1

 The least number of rectangles/squares formed is 1

Run Again (Y/N): Y
```

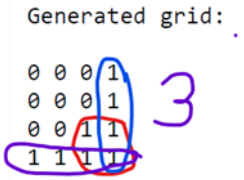


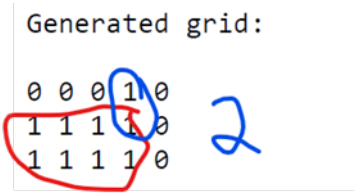


```
Enter the number of rows (2-10): 4
Enter the number of cols (2-10): 4


 Generated grid:

 1 0 1 0
 0 1 0 0
 0 0 0 0
 0 0 0 1

 The least number of rectangles/squares formed is 4

Run Again (Y/N): y

Enter the number of rows (2-10): 4
Enter the number of cols (2-10): 4

 Generated grid:

 1 1 0 1
 0 0 0 0
 0 0 0 0
 1 0 0 1

 The least number of rectangles/squares formed is 2

Run Again (Y/N): Y

Enter the number of rows (2-10): 4
Enter the number of cols (2-10): 4

 Generated grid:

 0 0 0 1
 0 0 0 1
 0 0 1 1
 1 1 1 1

 The least number of rectangles/squares formed is 3

Run Again (Y/N): y

Enter the number of rows (2-10): 3
Enter the number of cols (2-10): 5

 Generated grid:

 0 0 0 1 0
 1 1 1 1 0
 1 1 1 1 0

 The least number of rectangles/squares formed is 2

Run Again (Y/N): N
```









Name the program: `MinRectanglesXX.java` or `MinRectanglesXX.cpp`, where XX are your initials.

2. Write a program that given a full scorecard calculates the final score of a bowling match. The final score is the sum of all 10 frames according to the following standard rules of bowling:

Strike (Denoted as X):

- If all 10 pins are knocked down in the first shot of a frame, a strike is earned. No further shot is rolled for the current frame. A strike earns 10 points for the frame, plus the sum of the pins knocked down in the next two shots in following frames.

Spare (Denoted as /):

- If all 10 pins are knocked down using both shots of a frame, a spare is earned. A spare earns 10 points for the frame, plus the number of pins knocked down in the first shot of the next frame.

Open Frame:

- If all 10 pins are not knocked down using both shots of the frame (9 or fewer pins knocked down), an open frame is earned. An open frame only earns the number of pins knocked down in that frame.

The 10th Frame:

The 10th frame is slightly different:

- If you roll a strike in the first shot of the 10th frame, 2 more shots are immediately given.
- If you roll a spare in the first two shots of the 10th frame, 1 more shot is immediately given.
- If you leave the 10th frame open after two shots, the game is over, and no additional shot is given.

The score for the 10th frame is the total number of pins knocked down for all the rolls in the 10th frame.

The input from a data file starts with a line containing the number of test games as integer N between [1,10]. The following N lines consist of a single player's full score card from a complete game of bowling. An X represents a strike, and a / represents a spare. All inputs will be valid complete games. For each test case, output to the screen the final score, complete with pin count and in a cumulative graphical ASCII format. For a throw that knocks down no pins, a dash (-) is used instead of a '0'.
Use any appropriate data structure. Let the user enter the file name from the keyboard. Refer to the sample output below.

Sample File:

```
4
9/X9/X9/X9/X9/9/9
XXXXXXXXXXXX
90909090909090909090
81092/X637052X062/X
```

Sample Run:

```
Enter the file name: bowling.txt

Game        1    2    3    4    5    6   7    8    9     10
-------------------------------------------------------------
|       | 9|/|  |X| 9|/|  |X| 9|/|  |X| 9|/|  |X| 9|/| 9|/|9|
|       |----|----|----|----|----|----|----|----|----|------|
|Game 1 |  20|  40|  60|  80| 100| 120| 140| 160| 179|   198|
|-----------------------------------------------------------|
|       |  |X|  |X|  |X|  |X|  |X|  |X|  |X|  |X|  |X| X|X|X|
|       |----|----|----|----|----|----|----|----|----|------|
|Game 2 |  30|  60|  90| 120| 150| 180| 210| 240| 270|   300|
|-----------------------------------------------------------|
|       | 9|-| 9|-| 9|-| 9|-| 9|-| 9|-| 9|-| 9|-| 9|-| 9|-| |
|       |----|----|----|----|----|----|----|----|----|------|
|Game 3 |   9|  18|  27|  36|  45|  54|  63|  72|  81|   90|
|-----------------------------------------------------------|
|       | 8|1| -|9| 2|/|  |X| 6|3| 7|-| 5|2|  |X| -|6| 2|/|X|
|       |----|----|----|----|----|----|----|----|----|------|
|Game 4 |   9|  18|  38|  57|  66|  73|  80|  96| 102|   122|
-------------------------------------------------------------
```

Name the program: `BowlingXX.java` or `BowlingXX.cpp`, where `XX` are your initials.

3. A ping pong tournament is about to begin, and the tournament is supposed to be in a round robin format between `n` teams. In a round robin format, every team plays every other team exactly once. Write a program to validate that every team plays every other team once. Input will be from a data file. The first line of input consists of a single integer, `t`, in the range of `[1,10]` indicating the number of schedules. Each schedule is comprised of the following:

- The first line consists of two space-separated integers, `n` and `m`, where `n` represents the number of teams `[1,30]` in the round robin and `m` represents the number of games `[1,400]`.
- The next `n` lines contain team names, which are strings of one or more words.
- The next `m` lines represent the `m` games. Each game is in the format `X v Y`, where `X` and `Y` are the indexes of two teams that are playing each other. Indices begin at 1 (team numbers are not zero-indexed).

Output to the screen each schedule:

- If the schedule is in a round robin format, print "`YES`"; otherwise, print "`NO`", followed by a newline.
- If there are invalid games (a team playing itself or playing a team that doesn't exist, print "`INVALID GAMES`".
- If there are teams that aren't playing each other in the schedule, print "`MISSING GAMES`" followed by a newline, then followed by the missing games. Print the games in the form `A v B`, where `A` is the team, whose name comes first alphabetically. List in order the missing games alphabetically and separate each game with a newline.
- If there are teams which play each other more than once, print "`DUPLICATE GAMES`" followed

by a newline, then followed by the duplicate match. If two teams play each other more than twice, only print the duplicate game once. Print the games in the same format as the bullet listed above and separate each game with a newline.

Let the user input the file name from the keyboard. Use any appropriate data structure. Refer to the sample output below.

Sample File:

```
3
3 3
University of Texas
Texas A&M
Rice University
1 v 3
3 v 2
2 v 1
4 5
Harvard
Princeton
Cornell
Dartmouth
1 v 2
3 v 1
4 v 3
2 v 1
1 v 4
1 1
Lonely Guy
1 v 1
```

Sample Run:

```
Enter file name: robin.txt

Schedule 1:
YES

Schedule 2:
NO
MISSING GAMES
Cornell v Princeton
Dartmouth v Princeton
DUPLICATE GAMES
Harvard v Princeton

Schedule 3:
NO
INVALID GAMES
```

Name the program: `RoundRobinXX.java` or `RoundRobinXX.cpp`, where `XX` are your initials.

Required Problem- Comprehensive.

4 (**). Write a program that plays the casino game blackjack, or 21.

General Rules

Blackjack is a card game in which the object is to get a hand that is as close as possible to 21, without going over. A player starts out with two cards and can request more cards, one at a time. Numeric cards have their numeric value, and face cards have value 10. An ace has the value 1 or 11, which ever value is more advantageous to the user. For example, a hand with two aces and one five has the value of `11 + 1 + 5 = 17`. In this case, the second ace is counted as a value of 1, since if it were counted 11 the hand value would be 27. A two-card hand with an ace and 10 is always counted as 21 and is called a "blackjack" hand.

Basic Game Play

At beginning, the program welcomes the user and asks the user to select one of two modes:

- Game mode- Card values are randomly generated. The user of the program is the player and the program acts as the dealer. This is the default mode.
- Demo mode- Card values are entered by the user for computer and player. This mode is used for testing so modify game play as stated below.

Next two cards are dealt each to the player and the dealer. The values of player's cards are shown. The value of one of the dealer's cards is shown, with the other value hidden until the hand is over. The hidden dealer card is called the "hole" card. Once the player's hand is dealt, the player inputs the following one-character commands, depending on the player's status. Note all character inputs are in capital letters (check case) and not all commands are always available:

Command	Character	Description
Hit	H	Take another card
Stand	S	Stand on hand, proceed to dealer play
Double Down	D	Double the bet
Splitting Pairs	P	Split cards into two hands
Insurance	I	Check against dealer blackjack
Quit Hand	Q	Quit hand
Continue Playing	A	Proceed to next hand

If the player's hand value goes above 21, the player "busts" and loses the hand. The dealer reveals the hole card but does take any further cards. The hand then concludes. If the player does not bust, or the player's hand reaches a value of 21, then the dealer plays. This is done by the dealer taking cards until the dealer's hand value is 17 or higher, or the dealer busts. If the dealer's hand has a value of 17 and the hand includes an ace, then the dealer must take one more card. At the conclusion of dealer play, all dealer cards are revealed, including the hole card.

If neither the player nor dealer busts, then the winning hand is the one with the higher value. If the hand values are the same, then the hand is a tie, which is called a "push". A two-card blackjack hand beats a hand of value 21 with more than two cards. During game play, cards are displayed in the form shown in the examples below. The dealer's hole card is displayed as `'X'`. When the player's hand value reaches 21, the player is not prompted for a command, and the dealer's play commences automatically. If the player uses the `'Q'` command, the player loses immediately. The dealer does not play, and the hole card is not revealed. Here is a basic run of the program:

```
Welcome to Blackjack!!!

Do you wish to play in (E)-Demo or (G)-Game mode: G

Player's Hand: [13] 3S, JC
Dealer's Hand: [??] 5H, XX

(H)-Hit (S)-Stand: H

Player's Hand: [22] 3S, JC, 9H
Dealer's Hand:  [7] 5H, 2D

Player busts.
```

```
Enter (Q) to quit; Play another hand (A): A

Player’s Hand: [18] KS, 8D
Dealer’s Hand: [??] 9C, XX

(H)-Hit (S)-Stand: S

Player’s Hand: [18] KS, 8D
Dealer’s Hand: [22] 9C, 3H, 10H

Dealer busts.

Enter (Q) to quit; Play another hand (A): A

Player’s Hand:  [6] 2H, 4C
Dealer’s Hand: [??] QH, XX

(H)-Hit (S)-Stand: H

Player’s Hand: [16] 2H, 4C, 10C
Dealer’s Hand: [??] QH, XX

(H)-Hit (S)-Stand: H

Player’s Hand: [19] 2H, 4C, 10C, 3D
Dealer’s Hand: [??] QH, XX

(H)-Hit (S)-Stand: S

Player’s Hand: [19] 2H, 4C, 10C, 3D
Dealer’s Hand: [20] QH, JS

Dealer wins.

Enter (Q) to quit; Play another hand (A): A

Player’s Hand: [12] 8C, 4S
Dealer’s Hand: [??] 6D, XX

(H)-Hit (S)-Stand: H

Player’s Hand: [19] 8C, 4S, 7C
Dealer’s Hand: [??] 6D, XX

(H)-Hit (S)-Stand: S

Player’s Hand: [19] 8C, 4S, 7C
Dealer’s Hand: [19] 6D, 6S, 7D

Push.

Enter (Q) to quit; Play another hand (A): Q
```

Other features to be included:

Reshuffling

The reshuffling feature of the program detects when 52 cards have been dealt and continues play uninterrupted by restarting dealing with a new shuffled deck of 52 cards. Whenever reshuffling is needed, it happens automatically. The user is informed when reshuffling happens with the message `"Reshuffling ..."`. After the message, play continues where the program left off. Here is an example of reshuffling during a hand:

```
Player’s Hand:  [6] 2H, 4C
Dealer’s Hand: [??] QH, XX

(H)-Hit (S)-Stand: H

Player’s Hand: [16] 2H, 4C, 10C
Dealer’s Hand: [??] QH, XX

(H)-Hit (S)-Stand: H

Reshuffling...

Player’s Hand: [19] 2H, 4C, 10C, 3D
Dealer’s Hand: [??] QH, XX

(H)-Hit (S)-Stand: S

Player’s Hand: [19] 2H, 4C, 10C, 3D
Dealer’s Hand: [20] QH, JS

Dealer wins.

Enter (Q) to quit; Play another hand (A): Q
```

Betting

The betting feature of the program allows the user to enter a bet before each hand. During the game, the program keeps a running total of the player’s winnings. The total is incremented or decremented after each hand, depending on whether the player wins or loses. An initial value of $1000 is given. The minimum bet is $10, and the player can only bet in $10 increments up to the total amount in the player’s current hand. Error check. When a hand is finished, the player’s total winnings are printed. The game should quit automatically when the player runs out of money. The following table defines how winnings are computed for a hand:

Hand Outcomes	Win/Loss Amount
player wins regular hand	bet
dealer wins regular hand	-bet
player wins with blackjack hand	1.5 * bet
player and dealer tie	0
player wins doubled down hand	2 * bet
player wins doubled down hand with blackjack	3 * bet
player wins both split hands	2 * bet
player loses both split hands	-2 * bet
player wins one, loses one split hands	0
player win one, ties one split hands	bet
player loses one, tie one split hands	-bet

Here is sample play with betting:

```
Current total: $1000
Bets must be made in increments of 10. (Ex - 10, 20, 30, etc.)
Minimum bet: 10- Maximum bet: 1000

Place your bet: 50

Player's Hand: [13] 3S, JC
Dealer's Hand: [??] 5H, XX

(H)-Hit (S)-Stand: H

Player's Hand: [22] 3S, JC, 9H
Dealer's Hand:  [7] 5H, 2D

Player busts.

Current Winnings: 950

Enter (Q) to quit; Play another hand (A): A

Place your bet: 10

Player's Hand: [18] KS, 8D
Dealer's Hand: [??] 9C, XX

(H)-Hit (S)-Stand: S

Player's Hand: [18] KS, 8D
Dealer's Hand: [22] 9C, 3H, 10H

Dealer busts.

Current Winnings: 960

Enter (Q) to quit; Play another hand (A): Q
```

Insurance

When the dealer's face up card is an ace, a side bet of up to half of the original bet, can be made that the dealer's hole card is a face card insuring against a blackjack for the dealer. If the dealer's card is a

face card, it is turned up, and the player who made the insurance bet wins and paid double the amount of their half bet a 2 to 1 payoff. If the dealer does not have a blackjack, the player loses insurance bet and can still play a normal hand. If the player has blackjack as well- it is a push. For example, if the player's original bet was $10 the player pays as much as $5 for insurance. If the dealer has a blackjack, the player is paid 2-1 money for the insurance bet. So, if the player bet $5 they would receive $10. However, if the player has a blackjack as well the player will still lose the original bet, breaking even overall.

```
Player's Hand: [17] 7S, KD
Dealer's Hand: [??] AC, XX

(H)-Hit (S)-Stand (I)-Insurance:
```

Doubling Down

Another option open to the player is doubling the bet when the original two cards dealt total 9, 10, or 11. The player places a bet equal to the original bet and is given just one more card. This command is only available immediately after the player sees the initial deal, and the initial deal is not a blackjack hand. With two fives, the player may split a pair, double down, or just play the hand in the regular way.

```
Place your bet: 10

Player's Hand: [10] 3S, 7C
Dealer's Hand: [??] 8C, XX

(H)-Hit (S)-Stand (D)-Double Down: D

Bet doubled to 20
```

Splitting Pairs

When the player has a hand with two of the same cards, the player may choose to split the hands into two separate hands. To do this, the use the command character 'P', which indicates that the player wants to split. This command is only available as the first player command in a hand. That is, splitting can only happen immediately after the hand is dealt, and the hand has two cards of the same value. Splitting is allowed on hands with two different face cards, e.g., a Jack and a Queen, since these both have a value of 10. Play on the separate hands proceeds individually, in the same way as for a single hand. The first hand is played to conclusion, then the second hand, then the dealer's hand if necessary. The precise play of the hand goes like this:

- If the first hand busts, the message "`Player busts`" is output, and play proceeds to the second player hand.
- If the second hand also busts, the message "`Player busts`" is output again, and play concludes by printing all three hands, with the dealer hole card revealed
- If one or both player hands does not bust, then the dealer plays, and two messages are output indicating the conclusion of each of the split hands.

```
Player's Hand:  [4] 2C, 2S
Dealer's Hand: [??] 5D, XX
```

```
(H)-Hit (S)-Stand (P)-Split: P

Player's First Hand:   [4] 2C, 4H
Player's Second Hand:  [6] 2S, 4D
Dealer's Hand:        [??] 5D, XX

Enter command for first hand (H)-Hit (S)-Stand: H

Player's First Hand:  [10] 2C, 4H, 4S
Player's Second Hand:  [6] 2S, 4D
Dealer's Hand:        [??] 5D, XX

Enter command for first hand (H)-Hit (S)-Stand: H

Player's First Hand:  [18] 2C, 4H, 4S, 8C
Player's Second Hand:  [6] 2S, 4D
Dealer's Hand:        [??] 5D, XX

Enter command for first hand (H)-Hit (S)-Stand: S

Player's First Hand:  [18] 2C, 4H, 4S, 8C
Player's Second Hand:  [6] 2S, 4D
Dealer's Hand:        [??] 5D, XX

Enter command for second hand (H)-Hit (S)-Stand: H

Player's First Hand:  [18] 2C, 4H, 4S, 8C
Player's Second Hand: [16] 2S, 4D, 10C
Dealer's Hand:        [??] 5D, XX

Enter command for second hand (H)-Hit (S)-Stand: H

Player's First Hand:  [18] 2C, 4H, 4S, 8C
Player's Second Hand: [25] 2S, 4D, 10C, 9D
Dealer's Hand:        [??] 5D, XX

Player busts.

Player's First Hand:  [18] 2C, 4H, 4S, 8C
Player's Second Hand: [25] 2S, 4D, 10C, 9D
Dealer's Hand:        [17] 5D, 9C, 3S

Player wins first hand. Dealer wins second hand.

Enter (Q) to quit; Play another hand (A): Q
```

Use user-defined functions/methods in your program and any appropriate data structures. Output should be user friendly.

Name the program: `BlackjackXX.java` or `BlackjackXX.cpp`, where XX are your initials.

Extra Credit: Implement the following problem. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points)

Wordle is a word game in which players have six attempts to guess a five-letter word, with feedback given for each guess in the form of colored tiles indicating when letters match or occupy the correct position. Therefore, write a similar program that allows the user to compare a target word and an attempted guess. The program will compare the two words and provide the indicated output. Each word will be a 5-letter word consisting of lower-case letters only, and each word will contain five unique lower-case letters. The program should compare the two words letter-by-letter using the following rules:

- If the letter in position `N` of the guess matches the letter in position `N` of the target, the upper-case form of the letter will be printed in position `N` of the output.
- If the letter in position `N` of the guess is not anywhere in the target, an asterisk will be printed in position `N` of the output.
- If the letter in position `N` of the guess is in the target, but not in position `N`, the lower-case form of that letter in guess will be printed in position `N` of the output.

The input will be from a data file. The first line is a number `N`, representing the number of lines of data to follow. `N` will be in the range of `[1,50]`. The next `N` lines of data consist of two five-character strings each consisting only of lower-case letters and one space separating the words. For each pair of words, output to the screen the five characters consisting of uppercase letters, lowercase letters, and asterisks, properly labeled. Let the user enter the file name from the keyboard. Use any appropriate data structure. Refer to the sample output below.

Sample File:

```
5
logic texas
purse purse
china cloth
shore ascot
abcde edcba
```

Sample Run:

```
Enter file name: wordletest.txt

Pair 1: *****
Pair 2: PURSE
Pair 3: C***h
Pair 4: *s*o*
Pair 5: edCba
```

Name the program: `WordleXX.java` or `WordleXX.cpp`, where `XX` are your initials.