CS1102: Data Structures and Algorithms

Tutorial 4 Stacks and Queues

Week of 22/02/10

1. The following two classes are available to you. The `Stack` class implements a stack ADT, while the `Queue` class implements a queue ADT.

```
class Stack<E> {
  // Data members are private and are not shown

  public boolean isEmpty() { ... }
  public int size() { ... }
  public void push{E item) { ... }
  public E pop() { ... }
  public E peek() { ... }
}

class Queue<E> {
  // Data members are private and are not shown

  public boolean isEmpty() { ... }
  public int size() { ... }
  public void enqueue(E item) { ... }
  public E dequeue() { ... }
  public E getFront() { ... }
}
```

   Draw diagrams representing the contents of stack `s1`, stack `s2` and queue `q` at the end of the following program:

```
Queue<Integer> q = new Queue<Integer>();
Stack<Integer> s1 = new Stack<Integer>();
Stack<Integer> s2 = new Stack<Integer>();

s1.push(new Integer(3));
s1.push(new Integer(2));
s1.push(new Integer(1));

while(!s1.isEmpty()) {
  s2.push(s1.pop());
  if(!s1.isEmpty()) s2.push(s1.peek());
  q.enqueue(s2.peek());
}

s1.push(q.dequeue());
```

CS1102: Data Structures and Algorithms

2. Given a 2D array of integers (`data`) with m rows and n columns, explain how you would check whether there is a path of adjacent 1's from the first element (`data[0][0]`) to the last element (`data[m-1][n-1]`). We consider two 1's to be adjacent if they are next to each other in the same column or in the same row. You should use stack(s) in your solution. If a path exists, you should also explain how to output the path.

   Hint: You will need to use one stack for storing the path that has been traced so far and a list to store the visited element.

m = 5, n = 7

	0	1	2	3	4	5	6
0	1	1	1	0	1	1	1
1	1	0	1	1	1	0	1
2	0	0	0	0	0	1	1
3	0	1	1	1	0	1	0
4	1	1	0	0	0	1	1

path of 1's exists

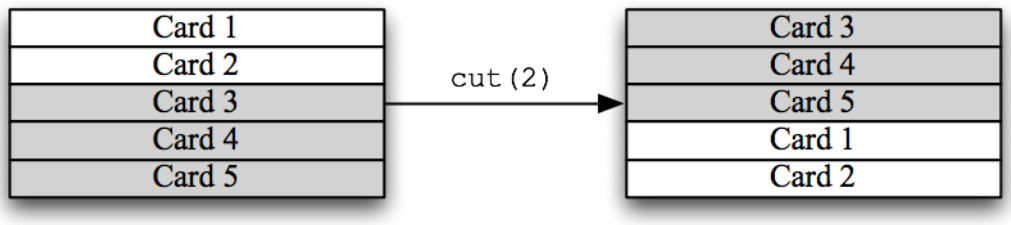
	0	1	2	3	4	5	6
0	1	1	0	1	1	1	1
1	0	1	1	1	0	0	1
2	1	1	0	1	1	1	1
3	0	1	1	1	1	1	0
4	1	1	0	0	0	0	1

path of 1's does not exist

3. The following classes are available to you. The `Card` class represents a single playing card, while the `Deck` class represents a complete deck of 52 cards.

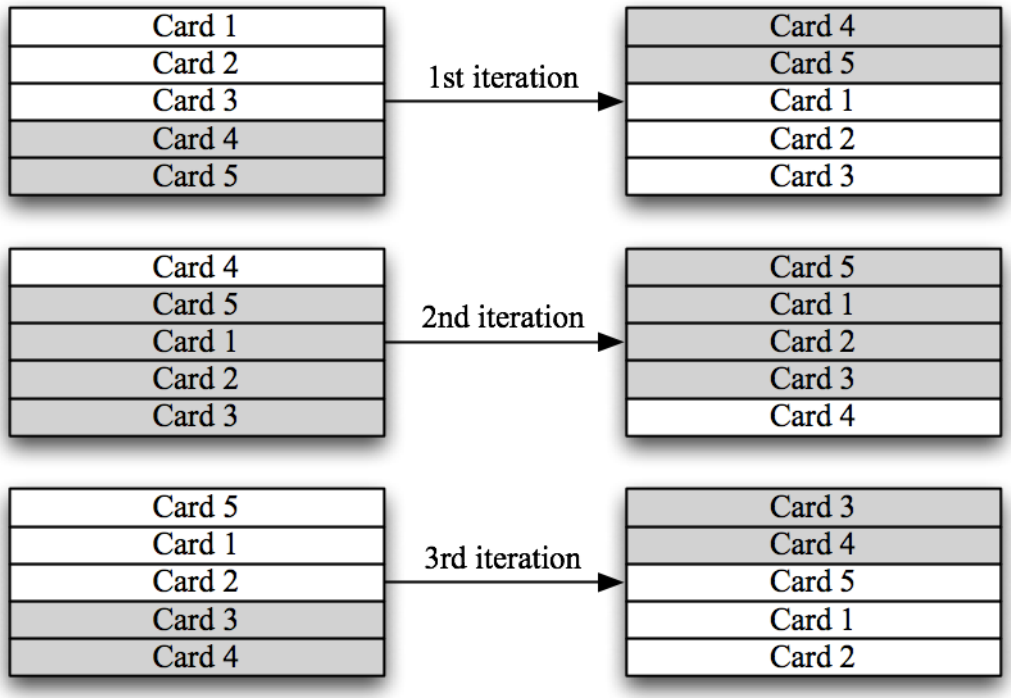
```
class Card {
  private String suit;
  private String denomination;

  public Card(String suit, String denomination) { ... }
  public String getSuit() { ... }
  public String getDenomination() { ... }
  public void setSuit(String suit) { ... }
  public void setDenomination(String denomination) { ... }
}

class Deck {
  public static final int NUM_OF_CARDS = 52;

  private Stack<Card> cards; // cards represent a deck of cards
                             // stacked face down on the table.
                             // In other words, cards.pop() will
                             // remove the top most card from the
                             // deck.

  public Deck() { ... } // The constructor initializes the stack
                        // with all 52 cards, but in some
                        // random order.
}
```

   a. Write an instance method `public void cut(int numOfCutCards)` for the `Deck` class that simulates cutting the deck of cards. Cutting a deck of cards is done by taking the top `numOfCutCards` cards from the deck, placing them on the table, then placing the remaining bottom portion of the deck on top of the `numOfCutCards` cards. You may not use arrays in your method. You can assume that the deck will always have 52 cards and `numOfCutCards` will always be less than 52. The following diagram illustrates calling `cut(2)` on a stack of 5 cards:



   b. Write another instance method `public void shuffle(int numOfIterations)` for the `Deck` class that simulates shuffling the deck of cards. You should assume that shuffling is done by moving a *random*-sized stack of cards from the bottom of the deck to the top and

and repeating the process for a total of `numOfIterations` times. The following diagram illustrates calling `shuffle(3)` on a stack of 5 cards:



c. Another way of shuffling is called the riffle or dovetail shuffle. In this shuffle, half of the deck is held in each hand with the thumbs inward, and then the thumbs release the cards so that they fall to the table interleaved (http://en.wikipedia.org/wiki/Shuffling#Riffle). Briefly explain how you would implement such a shuffle for the `Deck` class.

4. The language Lisp, each of the four basic arithmetic operators appears before an arbitrary number of operands, which are separated by spaces. The resulting expressions are enclosed in parentheses. The operators behave as follows:
    a. (+ a b c …) returns the sum of all the operands, and (+) returns 0.
    b. (- a b c …) returns a-b-c-… and (-a) returns –a. The minus operator must have at least one operand.
    c. (* a b c …) returns the product of all the operands, and (*) returns 1.
    d. (/ a b c …) returns a/b/c/… and (/a) returns 1/a. The divide operator must have at least one operand.

You can form larger arithmetic expressions by combining these basic expression using a fully parenthesized prefix notation. For example, the following is a valid Lisp expression:

(+(-6) (*2 3 4))

The expression is evaluated successively as follows:

(+ -6 (*2 3 4))
(+ -6 24)
18

Design and implement a recursive algorithm that uses a stack to evaluate a legal Lisp expression composed of the four basic operators and integer values.