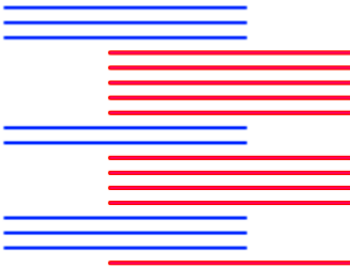# BREAKTHROUGH!
Theory Questions

These questions refer to the **Preliminary Material** and the **Skeleton Program**,
but **do not** require any additional programming

TOTAL MARKS: 80

---

1   Examine the private method **MoveCard**. Currently this method returns the local variable **Score.**

(a) State a more appropriate name for this local variable.   [1]

(b) Currently the **MoveCard** method returns an integer which represents the score for the card that was moved. Sometimes this return value is ignored.

Evaluate the choice (of the programmer) to ignore the return value in some cases and to return 0 in some cases, and suggest an alternative implementation.   [4]

2   The class **CardCollection** currently contains an interface that exposes the internal data structure of a list. For the sequence and the discard pile, a more appropriate data structure would be either a queue or a stack.

(a) Justify whether you would use a queue or a stack. When giving your answer, link the functionality of the data structure to the behaviour of the game.   [4]

(b) In order to implement a stack or a queue for the sequence, justify any changes (or lack thereof) that you would make to the inheritance structure.   [2]

(c) How would creating a new class to handle a **CardCollection** that uses a stack or a queue improve encapsulation?   [2]

3   The **Shuffle** method of the **CardCollection** class currently swaps 10,000 pairs of randomly selected cards in order to shuffle the deck.

Another way of shuffling the deck is to use a method that humans would normally use called a 'riffle shuffle'. This involves splitting the deck into two approximately even piles and then flicking through each pile from the bottom while combining the halves together into a single deck. Another way of thinking of this would be to imagine pushing the two halves together and having a random number of cards between each card from each half as they recombine.

For example, a deck combined from a blue half and an orange half might look something like this:

Note that in the perfect case a riffle shuffle would use one card from each half, but of course this is not desired, and in reality, between 0 and 5 cards will normally interspace the cards from the other half at any time.

a) Write a detailed algorithm for riffle shuffle in any format you choose (e.g. structured English, pseudocode, flow chart).   [6]

b) Explain the space complexity of your algorithm.   [1]

4 Examine the **CheckIfLockChallengeMet** method of the **Breakthrough** class and the **CheckIfConditionMet** method of the Lock class.

```
Lock:
Challenge Met: P c, F c, K c
Not met: P a, F a, P a
Sequence: P c, F c, K c, P a, F a, P a
```

(a) For the above sequence and lock, complete a trace table like the one below for the **CheckIfLockChallengeMet** method of the **Breakthrough** class. **[5]**

Count	SequenceAsString	Return value
	""	
5		

Note: you might not require all of the rows

(b) If the above lock had a third challenge as below, then how would the trace table change? (Complete an updated trace table) [3]

```
Not met: F a, P a
```

Count	SequenceAsString	Return value
	""	
5		

Note: you might not require all of the rows

5 Examine the **ProcessLockSolved** method in the **Breakthrough** class and any necessary methods called by that method.

(a) When a new lock is set, if that lock has been solved before, it will still appear before being automatically replaced with a new lock the following turn (and treated as if the player had just solved the first new lock) but reward the player for solving the lock. Explain why. [2]

(b) Describe the logical change you would make to the code (no need to write the actual code, although you can) to ensure that this no longer happens. [2]

6 Examine the **Shuffle** method in **CardCollection**. This method will make 10,000 random swaps of cards in the deck.

(a) Explain how the effectiveness and efficiency of this algorithm decreases as the number of cards in the deck reduces. [3]

(b) Other than introducing a riffle shuffle, justify how you could improve the effectiveness and efficiency of the algorithm by describing any changes below. [2]

7 **ToolCards** can be instantiated with either two or three arguments.

(a) Explain what happens in the case where a third argument is supplied compared to the case where only two arguments are supplied. [3]

(b) State the purpose of a constructor. [1]

8 Examine the classes **Card**, **ToolCard** and **DifficultyCard**.

(a) Using evidence from these classes in the program, explain the difference between an abstract and a concrete class. [4]

(b) Using evidence from the **Card** method, explain the difference between a class variable (static) and an attribute. [2]

9 Find an example in the code for each of the following. Only write out the one or two relevant line/s of code.

(a) Inheritance [1]

(b) Aggregation association [1]

(c) A dynamic data structure [1]

10 This question refers to the concept of polymorphism and how it is used in the skeleton code.

(a) Choose and then write out one or more lines of the skeleton program which demonstrate polymorphism and justify why this is an example of polymorphism. [4]

(b) Define the term 'polymorphism'. [2]

11 A suggestion has been made to introduce a new **AdvancedLock** that has a secret final challenge which is only revealed once the basic challenges have been solved.

Explain the steps that you would take in order to do this, i.e. the logical nature of the change/addition and the reason for each step.

You are not required to implement this or to write any actual code. [6]

12 Examine the Process method in the **DifficultyCard** class and the **PlayCardToSequence** and **GetCardFromDeck** methods of the **Breakthrough** class.

Using the scenario below:

```
Not met: P a, F a, K a
Sequence: P a, F a
Hand: P b, K a, F b, K c, P a
```

The player plays the 'K a' card to the sequence and then draws a difficulty card which will require them to either discard a key or five cards from the deck. The player would like to discard the 'K c' from their hand, which is currently in position 4.

Explain what will happen when the **Process** method is called under these circumstances including specific references to the lines of code executed and in which order as well as the values of variables, especially **ChoiceAsInteger**.

You will need to ensure that you look at the **PlayCardToSequence** and **GetCardFromDeck** methods in **Breakthrough** to be certain of the state of the Hand and Sequence at the moment the **DifficultyCard** is drawn. [8]

13 The terms 'HAND', 'SEQUENCE', 'DECK' and 'DISCARD' all appear at least once in the code and in some cases more than once. This is an example of hard-coded values which make code difficult to maintain and understand and also make it more prone to errors.

    (a) Describe one method of avoiding hard-coding values that makes code easier to maintain. [2]

    (b) Explain why using hard-coded values makes the code more prone to errors and harder to understand. [2]

14 Exception handling is used in several places in the skeleton code; two examples of this involve the use of file handling.

    (a) Describe why it is important to always use exception handling when dealing with files. [2]

    (b) Give an example of another situation (not file handling) where exception handling is useful (it doesn't have to be from the skeleton code) and explain why. [2]

15 This question refers to the **PlayGame** method of the **Breakthrough** class.

Explain the use of the private attribute **GameOver** in this method, specifically explaining how it is set and why it is used as the condition for two iterative statements. [2]

END OF QUESTIONS