PROBLEM ONE: MOVIE RATING PREDICTIONS

The Distance Formula is given by the following formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

This formula is commonly used to find the Cartesian distance between two points in the real world, perhaps the distance between two flying aircraft. However, this formula can also be applied to problems that are more abstract than the real world. For instance, the distance formula can be applied to help predict what movies a person might like.

The following is a table of movie reviews from 4 Professional Reviewers. Each reviewer has rated six movies where the movies are rated from 1 (terrible) to 5 (excellent).

	1) Inception	2) Toy Story 3	3) Scott Pilgrim vs. the World	4) Harry Potter	5) TRON: Legacy	6) Easy A
1) Jason	3	1	5	2	1	5
2) David	4	2	1	4	2	4
3) Keenan	3	1	2	4	4	1
4) Derek	5	1	4	2	4	2

Write a program where this data is stored in an array or other data structure. Your program must prompt the user to enter their personal ratings for the first three movies (Inception, Toy Story 3, and Scott Pilgrim vs. the World). It is assumed that the users have seen those movies. Your program must then find the professional reviewer whose ratings most closely match the ratings input by the user. Search through each Professional Reviewer's ratings and find the smallest distance between between the users ratings and each Professional Reviewer's ratings for the first three movies. **You must use the distance formula as a metric and it must be implemented correctly**; apply the variables x, y and z to the first three movies. For example, if the user inputs a rating of 5 for Inception, your x variables should be set the following when calculating the distance between Jason's ratings and the user's ratings:

$$x_1 = Jason's\ review\ of\ Inception = 3 \text{ and } x_2 = User's\ review\ of\ Inception = 5$$

REQUIRED INPUT: Prompt User for Ratings of Inception, Toy Story 3, Scott Pilgrim
REQUIRED OUTPUT: The Professional Reviewer whose movie ratings are the mathematically smallest distance from the user's movie ratings.

PROBLEM TWO: A SINGLE INTEGER FOR TWO TEMPERATURES

Write a program that finds the temperature, as an integer, that is the same in both Celsius and Fahrenheit using the following formula:

$$Fahrenheit = \frac{9}{5}Celsius + 32$$

Your program must compute this integer value using the formula. The answer to this equation cannot be hard-coded anywhere in your program.

REQUIRED INPUT: None
REQUIRED OUTPUT: An integer number that represents a temperature that is the same in Celsius and Fahrenheit that was mathematically computed using the temperature conversion formula above.

PROBLEM THREE: EMIRPS

An emirp (prime spelled backwards) is a prime number whose reversal is also a prime. For example, 17 is a prime and 71 is also a prime, therefore 17 and 71 are emirps. Write a program that computes and displays the first 100 emirps. Display 10 numbers per line, with 3 spaces between each number and an empty line between each line. For example:

```
2    3    5    7    11    13    17    31    37    71

73    79    97    101    107    113    131    149    151    157

…
```

REQUIRED INPUT: None
REQUIRED INPUT: The first 100 emirps, computed. Display 10 numbers per line, with 3 spaces between each number and an empty line between each line.

PROBLEM FOUR: EIGHT QUEENS

Computing possible solutions to classic human games such as Chess as long been an application of Computer Science. The goal of the Eight Queens puzzle is to place eight queens on a chessboard such that no two queens can attack each other. That is, no two queens are on the same row, same column, or same diagonal. There are many possible solutions.

Your program must compute and display one possible solution to the Eight Queens puzzle on a 8x8 sized chessboard. Use the character 'O' to indicate empty positions, and 'X' to indicate positions with Queens on them. Your program must also include one space between each character.

Example:

```
X O O O O O O O
O O O O X O O O
O O O O O O O X
O O O O O X O O
O O X O O O O O
O O O O O O X O
O X O O O O O O
O O O X O O O O
```

REQUIRED INPUT: None
REQUIRED OUTPUT: A computed solution to the Eight Queens puzzle on an 8x8 chessboard. Use the character 'O' to indicate empty positions, and 'X' to indicate positions with Queens on them. Your program must also include one space between each character.

PROBLEM FIVE: ERROR CHECKING CODES

All Credit Card numbers follow certain patterns. A credit card must have between 13 and 16 digits and it must start with a 4, 5, 37, or 6.

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. This algorith is useful to determine if a card number is entered correctly or if a credit card is scanned correctly by a scanner. Almost all credit card numbers are generated following this validity check, which can be described as follows:

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number. For example, if a particular second digit were 4, then computing that digit would be $4 \times 2 = 8$. If a particular second digit were 6, then computing that digit would be $6 \times 2 = 12 \rightarrow 1 + 2 = 3$.

2. Now all all single-digit number from Step 1.

3. Add all digits in the odd places from right to left in the card number.

4. Sum the result from Step 2 and Step 3.

5. If the result from Step 4 **is** divisible by 10, the card number is valid. If the result from Step 4 **is not** divisible by 10, it is invalid.

Write a program that takes a credit card number as a long integer, entered as a command line arguement. Determine using Luhn's algorithm if the Credit Card number is valid or not.

Example valid Credit Card number: `4388576018410707`

Example invalid Credit Card number: `4388576018402625`

If you have an actually credit or a debit card, don't use the number for testing. We would like very much that your bank account remains secure, so please minimize the risk by keeping any actual credit or debit card numbers away from prying eyes. Use the example Credit Card numbers instead for testing.

REQUIRED INPUT: A Credit Card number between 13 and 16 digits, input as a command line argument.
REQUIRED OUTPUT: Compute the validity of the Credit Card number that was input using Luhn's algorithm and output the result of the validity test.

PROBLEM SIX: THE TIME POLICE AND PLAYFAIR'S CIPHER

Cryptography is the study and application of complex ciphers, used to pass messages through foreign and hostile settings. During World War I, the British used a cipher algorithm called Playfair's Cipher because it was reasonably fast to use and required no special equipment. It was later adopt by the Germans during World War II. Playfair's Cipher is no longer used in any real-world application because modern computer could easily break the cipher within seconds.

You are computer programmer from the early 1990's who has been selected by the Time Police to be sent back in time to fight the Nazis during World War II. Your are equipped with nothing but your IBM ThinkPad and a unending battery to provide the ThinkPad with power. You must write an algorithm to quickly translate intercept cryptographic message encoded using Playfair's Cipher.

Playfair's Cipher uses a 5x5 table containing a keyword or phrase. To generate the key table, fill in the spaces in the table with the letters of the keyword (dropping any duplicate letters), then fill the remaining spaces with the rest of the letters of the alphabet in order. The letter 'J' is combined into the letter 'I' onto one square. They key can be written in the top rows of the table, from left to right. The keyword together with the conventions for filling in the 5x5 table constitute the cipher key.

To encrypt a message, one would break the message into diagraphs (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. The two letters of the digraph are considered as the opposite corners of a rectangle in the key table. Note the relative position of the corners of this rectangle. Then apply the following 4 rules, in order, to each pair of letters in the plaintext:

1. If both letters are the same (or only one letter is left), add an "X" after the first letter. Encrypt the new pair and continue. Some variants of Playfair use "Q" instead of "X", but any uncommon monograph will do.
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row).
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column).
4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important – the first letter of the encrypted pair is the one that lies on the same **row** as the first letter of the plaintext pair.

To decrypt, use the INVERSE (opposite) of the first 3 rules, and the 4th as-is (dropping any extra "X"s (or "Q"s) that don't make sense in the final message when finished).

Write a program that uses the Playfair's Cipher algorithm to decode the following message using the keyword "**DASBOAT**":

KT XF XT YB OD BV XC BP FT AM
UT GD HP WD ST UA IB VT VS DU
EV CF KY IB RV OD BV

You may hard-code the message and keyword into your program, but it must be translated by your program using the algorithm above.

REQUIRED INPUT: None
REQUIRED OUTPUT: The deciphered message; decoded using Playfair's Cipher algorithm.

PROBLEM SEVEN: PIG LATIN

Pig Latin is a language game of alteration played in English. The purpose of the alteration is to both obfuscate the encoding and to indicate for the intended recipient the encoding as 'Pig Latin'.

For words that begin with consonants, move the leading consonant to the end of the word and add "ay". Thus, "town" becomes "owntay"; "keyboard" becomes "eyboardkay".

For words that begin with vowels, add "way" to the end. Thus, "ear" becomes "earway"; "ice" becomes "iceway".

Write a program that will parse a text file written in plain English and output the contents of the file translated into Pig Latin. Your program must also properly handle common written English punctuations including commas, quotation marks, colons, semicolons, apostrophes, periods, question marks and exclamation marks (you can assume that special characters not common to written English (i.e., not listed) will not have to be handled by your program).

Your program must accept the text file name as a command line argument.

Example Input:

> Wow! Did everybody see Watson on Jeopardy this week? It did very well. So well, that Ken Jenning's wrote a funny quip during Final Jeopardy: "I, for one welcome our new computer overlords."

Translated Pig Latin Output:

> Owway! Idday everybodyway eesay Atsonway onway Eopardyjay isthay eekway? Itway idday eryvay ellway. Osay ellway, atthay Enkay Enningjay's otewray away unnyfay ipquay uringday Inalfay Eopardyjay: "Iway, orfay oneway elcomeway ourway ewnay omputercay overlordsway."

REQUIRED INPUT: The text file name inputted as a command line argument.
REQUIRED OUTPUT: Output the translated text in Pig Latin, parsed by your program. Commas, quotation marks, colons, semicolons, apostrophes, periods, question marks and exclamation marks must be handled properly.

PROBLEM EIGHT: CHECKING SOURCE CODE

Most programming language code is full of different types of grouping symbols, such as;

- Parentheses: (and)
- Braces: { and }.
- Brackets: [and].

Note that grouping symbols cannot overlap. For example, (a{b)} is illegal.

Write a program to check whether a source-code file has valid group symbols, and if valid, output how many sets of each grouping sets were found in the source code. If the source code file is not valid, simply state the source code file is invalid. That is, if every grouping symbol is properly closed when open. You can check your program against its own source code. Pass the source code file name as a command line argument.

Example output for valid source code:

```
Number of Parentheses () sets: 20
Number of Braces {} sets: 13
Number of Brackets [] sets: 4
```

Hint: When parsing any group symbols, consider *stacking* them.

REQUIRED INPUT: The source-code file name as a command-line argument.
REQUIRED OUTPUT: If the program determines the source code file is valid, output how many sets of each grouping sets were found in the source code. If the source code file is not valid, simply state the source code file is invalid.

PROBLEM NINE: SECURITY AND PREVENTING SOCIAL ENGINEERING

Traditional password entry schemes are susceptible to “shoulder surfing” in which an attack watches an unsuspecting user enter their PIN number and uses it later to gain access to the account. One way to combat this problem is with a randomized challenge-response system. In these systems, the user enters different information every time based on a secret in response to a randomly generated challenge.

We will consider the following scheme in which the password consists of a five-digit PIN number (00000 to 99999). Each digit is assigned a random number that is 1, 2, or 3. The user enters the random number that correspond to their PIN instead of their actual PIN numbers.

For example, consider an actual PIN number 12345, To authenticate, the user would be presented with a screen such as:

```
PIN: 0   1   2   3   4   5   6   7   8   9
NUM: 3   2   3   1   1   3   2   2   1   3
```

The user would enter 23113 instead of 12345. This does not divulge the password even if an attack intercepts the entry because 23113 could correspond to other PIN numbers, such as 69440 or 70439. The next time a user logs in, a different sequence of random number would be generated.

Write a program to simulate the authentication process. The random numbers 1, 2 and 3 must be assigned by a randomly generated process in your program, not hard-coded. Output the random digits to the screen, input the response from the user, and output whether or not the user’s response correctly matches the PIN number. The program must accept the user’s actual PIN as a command line argument. The PIN must be five digits, and every digit must be unique, otherwise, your program should indicated a invalid PIN number was passed.

REQUIRED INPUT: A 5-digit PIN number as a command line argument, where every number is unique, including number 0-9. Then output the authentication screen and prompt the user to enter the coded PIN.
REQUIRED OUTPUT: A message indicating if the coded PIN was authenticated or not. If an invalid PIN number was passed via the command line argument, then output an invalid PIN message.