Department of Computer Science

COS132 2015
Practical 4

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Mr Madoda Nxumalo

0 Introduction

0.1 Uploads and Deadline

The uploads will become available on Monday 4 May 2015.

The submission deadline for this practical is **8 May 2015 at 20:00 PM**. Make sure that you have submitted your code to Fitchfork by then. **No late submissions will be accepted**.

0.2 Challenges

Note that the challenges are **NOT** part of assessed sections of the assignment. It is given as a challenge to those who find our usual assignments too trivial. It does not contribute to your marks. It is not expected from the Teaching Assistants to assist you to complete these assignments. It is intended for you to figure it out on your own and stimulate you to use the opportunities you have to develop your programming skills to the fullest. There will also not be any uploads on these.

0.3 Plagiarism Policy

All submissions for this practical and all future practical should be documented as specified in our plagiarism policy. Refer to the notes about our plagiarism policy which can be found on the COS132 website on ClickUP.

1 IsVowel [5 marks]

Write a value-returning function, `bool isVowel(char)`, that returns the value **true** if a given character is a vowel and otherwise returns **false**.

For this task, you are provided with the following files;

- **main.cpp**
- **vowels.h**
- **makefile**

vowels.h contains the `isVowel()` function. Notice how **vowels.h** is included in **main.cpp**.

You have to create a file called **vowels.cpp** containing the implementation of the `isVowel()` function.

The `isVowel` function must further be used in a program that prompts the user to input a string and outputs the number of vowels in the input string. The framework for this program is given in **main.cpp**. The input string may include spaces and punctuation. See page 149 in the textbook for guidance in how to deal with this.

The system containing all these files can be compiled with the following command:

```
g++ volwels.cpp main.cpp -o vowels.out
```

Notice that the h-file is not mentioned in the compile command, as it is already part of the code by being included in these cpp-files.

To run the program, type

```
./vowels.out
```

You are also given a makefile that can be used to compile and run the file. To use the makefile to compile the file, simply type the following command at the command prompt

```
make
```

To run the program with the makefile, type the following command.

```
make run
```

The makefile is written to automatically re-compile the program if you have made any changes to any of the files, before running it when this command is given. Include all your files along with the given makefile in both your uploads. Use the following command to create the tarball:

```
tar -cvz vowels.h vowels.cpp main.cpp makefile -f vowels.tar.gz
```

2 n Difference in Arrays Elements [8 marks]

Given an unsorted array of integers and a number n (all read from standard input).

Sort function

Implement a function with prototype `void sort(int array[], int s)` that sorts a given array in ascending order. In this prototype `array` is the array of integers and `s` is the size of the array.

Count pairs function

Write a function that can be used to know if it is possible to form pair of elements whose difference is n using only elements of the array. The function's return value must be the count of such pairs. The function prototype should be:

```
int countPairsDifference(const int array[], int s, int n);
```

`n` is the value difference, `s` is the size of the array `array`. This function should **NOT** assume that the given array is sorted. It may use the `sorter` function but should take care that it does not alter the content of the given array.

Example: Given $n = 4$ and $a[] = 7, 6, 23, 19, 10, 11, 9, 3, 15$, the function should return 6, as the pairs are: $(3, 7), (6, 10), (7, 11), (11, 15), (15, 19)$ and $(19, 23)$. Note that the pair (7, 3) is not one of the pairs complying with the specification as $3 - 7 = -4 \neq 4$.

Write a program that illustrates the use of these functions.

The following is a sample test run of a possible driver program (input is shown in bold).

```
Difference in Array Elements
Enter the number of array elements: 9
Enter the difference value: 4
Enter the sequence of elements: 7 6 23 19 10 11 9 3 15
Using these values, one can create 6 pairs with their elements having a difference of 4
The elements of the array in the given order is: 7 6 23 19 10 11 9 3 15
The elements of the array in sorted order is: 3 6 7 9 10 11 15 19 23
```

For this task you are given only a makefile. You have to create the following files required by the given makefile:

arrayDiffs.h	An h-file similar to the one given in task 1 containing the given function prototypes.
arrayDiffs.cpp	A cpp-file containing the implementations of the functions
main.cpp	The cpp-file that contains a main function that produces the output shown in the above test run.

Submission

You may assume the following:

- The size of the array will not exceed 500.
- The user will enter only valid input.

Include all your files along with the given makefile in your upload. Use a command similar to the one you used to create your tarball for task 1 to create a tarball that contains the required files.

Note that your driver is not evaluated. We use our own driver to test your functions with our own test suite.

Challenges

- Add input validation to ensure that the input array has only integers and has the correct number of integers.
- Implement and use a function that outputs the actual pairs.
- Think about ways in which you can improve the performance of the program, both in terms of memory usage and time to perform the operations.

3 Swap Letters [12 marks]

The essence of this assignment is to write a function that reverses character elements in an array. In order to achieve this, some supporting functions to initialise the array and to display the array is also needed.

Input

To enhance the testability of the function it is required that the array should be initialised using input that is saved in a file. The input file used for this purpose is a text file containing integer values. Initialisation of the character array is achieved by reading the integers from the file and converting them to the characters with the corresponding ASCII values.

The name of the input file need to be specified by the user. Furthermore, only characters within the ranges $(A \ldots Z)$ and $(a \ldots z)$ should be inserted in the array. If the input is out of this range, it should be ignored. The loop that reads form the file should should terminate if the end of the file is reached or if the input is not an integer.

Functions

You have to implement the following functions:

`void readElements(char arr[], int & size, ifstream & inF)`
To read the acceptable elements from the file and add them into the array.

`void swap(char & a, char & b)`
To swap two elements `a` and `b`, which are the arguments of the function.

`void reverseArray(char arr[], int size)`
To reverses the elements in the array.

`void print(char arr[], int size)`
To list the array elements in a single line on the standard output.

3.1 Testing

You are encouraged to write your own driver program to test your functions and also to create your own test data. The following is a sample test run of a possible driver program:

Array Element Reversal Function
Enter the file name: **chars.dat**
Array Elements Before Reversal: a b f D c A B
Array Elements After Reversal: B A c D f b a

It is assumed that the `reverseArray` function uses the `swap` function. Ideally one should also test the `swap` function alone before integrating its use in the `reverseArray` function

Submission

You may assume the following:

- The size of the array will not exceed 500.
- The user will enter only valid input.

You are provided with the makefile. You have to create these files; **main.cpp**, **arrayReverse.h** and **arrayReverse.cpp**. Include all your files along with the given makefile in your upload.

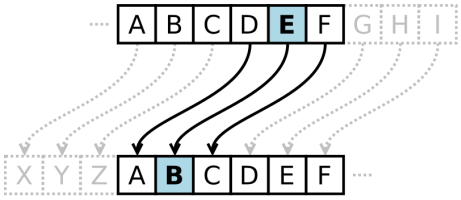
Note that your driver is not evaluated. We use our own driver to test your functions with our own test suite.

3.2 Challenges

- Implement the `swap` function without introducing any other variable, except the ones being exchanged.
- Implement the `readElements` function to be robust in terms of dirty data. If the file contains data that is not an integer, the file reader must continue reading values after the non-integer input. Reading these values must continue until the end of the file is reached and all acceptable elements in the file have been read.

4 Julius Caesar's Cipher [17 marks]

Julius Caesar's encryption and decryption technique is one of the simplest and most widely known encryption techniques. To apply it, each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, **E would become B**, and so on as shown in the following image by Matt_Crypto [Matt_Crypto(2013)]. The method is named after Julius Caesar, who used it in his private correspondence [Wikipedia(2015)].



Encryption: To encrypt a message, each alphabetical character in the message string has to be replaced with another character that is a defined integer value k letters from it in the alphabet. k may be positive or negative.
Decryption: To decrypt an encrypted message one has to reverse the encryption. This can be done by using the same function to shift the characters the same distance but in the opposite direction.

Implement the following functions that can be used both to encipher and to decipher messages;

- `char shift(int k, char letter)`

  k is the shift key and `letter` is a character. The function returns the character that is k positions from `letter` in the alphabet. If letter is not an alphabetical character, the function returns the given character. In other words, spaces and punctuation is not affected by encryption. The input characters as well as the characters returned, must belong to the alphabet, i.e. they have to be in the ranges $(A \ldots Z)$ and $(a \ldots z)$. When a shift value is specified that would result in a character that is beyond these ranges, the shift should proceed on the other end of the alphabet. For example, as illustrated in the output example below; the letter T shifts beyond z and thus the shift continues to $a, b..$ resulting to the letter C after encryption. The same applies when moving in the opposite direction. When decrypting b it moves past a to z and 8 more characters back to s. Capital letters should remain capital letters and lower case letters should remain lower case.

- `void cipher(int k, string & msg)` k is the shift key and `msg` is a string. The function alters the string by using the `shift` function with each characters in the string and replacing the character with the character returned by the function.

4.1 Testing

Write a driver program that prompts the user for a message and a key value. Allow the message to include spaces and punctuation. The program must then show the encrypted message that was created using the given key. To test if the function is working properly you should then call the `cipher` function with the encrypted message with the appropriate key to reverse the replacement. It should return the original message.

The following is a sample test run of this driver program:

The Caesar Encryption
Enter the list of characters to encrypt: **The me I see... is the me I'll be!**
Enter the shift key: 10
Encryption Output: Dro wo S coo... sc dro wo S'vv lo!
Decryption Output: The me I see... is the me I'll be!

Submission

For this task, you have to create your own files, captioned; **makefile**, **main.cpp**, **caesar.cpp** and **caesar.h**. They must be used in the same way as described in Task 1. These files must be included in your tarball submission.

Note that your driver is not evaluated. We use our own driver to test your functions with our own test suite.

5 Lo Shu Magic Squares [13 marks]

Write a program that reads in a 3×3 grid of integers from the user, then tells the user whether or not the grid is a Lo Shu Magic square.

The Lo Shu Magic Square is a grid with 3 rows and 3 columns. The Lo Shu Magic Square also has the following properties: [Kapoor and Kapoor(n.d.), NRICH team(n.d.)].

- The grid contains the numbers 1 through 9 exactly once.
- The sum of each row, each column, and each diagonal all add up to the same number. This is shown in **Table 1**.

Examples

Here are two examples of the program input and output;

```
The Lo Shu Square Magic
Enter the first row  values: 4 9 2
Enter the second row values: 3 5 7
Enter the last row  values: 8 1 6
The grid is: Lo Shu Square, and the sum is: 15
```

```
The Lo Shu Square Magic
Enter the first row  values: 4 9 2
Enter the second row values: 3 4 7
Enter the last row  values: 6 1 6
The grid is: NOT Lo Shu Square
```

			15
5	9	2	**16**
2	5	7	**14**
8	1	6	**15**
15	**15**	**15**	**16**

			15
4	9	2	**15**
3	5	7	**15**
8	1	6	**15**
15	**15**	**15**	**15**

Table 1: Left: not a Lo Shu square; Right: a Lo Shu Square)

Uploading

For this task, you have to create your own source code files. They must be used in a similar way as described in Task 1. All source files must be included in your submission uploads.

You **must** upload a `makefile` that will compile your program successfully similar to the examples in the previous tasks. Because of this, you may create and name your source code files however you wish.

Fitchfork also has certain requirements about makefiles:

1. The makefile must contain a `run` target which invokes `./yourprogram.out`, or whatever your executable is called.
2. The executable should be compiled/linked with the `-static` option,
   eg: `g++ -Wall main.cpp loShu.cpp -static -o magic.out`

As specific functions are not prescribed for this task, the functions are not tested separately. The program is assessed only as a unit.

Challenge: Extension of the Lo Shu Square

The above problem can be extended into a complete puzzle game, whereby the user is provided with only the (distinct) total sums of the rows, columns and diagonals as well as a few cells (for example; at most 3 cells) filled with integer values. In this case, the

sums may not necessarily be restricted to a specific constant. The user will then have to enter the remaining cell elements.

Should the user require to edit some of the cell values, then s/he can do so by specifying the cell row and column indices before editing. Afterwards the program should check whether the grid is a Lo Shu or Magic square.

Moreover, you can add functionality to allow the user to choose suitable two-dimension sizes (i.e. number rows and columns) of the magic square grid in order to increase the challenge of the game.

6 Driver's license Exam [15 marks]

The local Driver's License Office has asked you to write a program that grades the written portion of the license exam. The exam has a fixed number of multiple choice questions. The correct answers is saved on a file called **answers.dat**. The first value on the file is the number of answers. This is followed by the correct answers to the number of questions as specified.

The following table shows the correct answers for the 20 questions saved in the given file called **answers.dat**

1.	A	6.	B	11.	A	16.	C
2.	D	7.	A	12.	C	17.	C
3.	B	8.	B	13.	D	18.	A
4.	B	9.	C	14.	B	19.	D
5.	C	10.	D	15.	D	20.	B

Your program should read the file and store the correct answers in an array. It should ask the user to enter the student's answers for each of the questions. Store the answers in another array. After the student's answers have been entered, the program should display a message indicating whether the student passed or failed the exam. (A student must correctly answer 75% of the questions to pass the exam). It should then display the total number of correctly answered questions, the total number of incorrectly answered questions, and a list showing the question numbers of incorrectly answered questions.

Input validation:

- You may assume that the input file exist and contains clean and correct data.
- Only accept the letters *A, B, C or D* as answers. When a user enters an invalid value, an error message should be shown and the program must terminate immedeatly.

Output Example:

Driver's License Exam
Student: Enter your answers:
Question 1: **A**
Question 2: **D**
Question 3: **D**

...
Question 20: **B**
==========
The student have PASSED the test
The total number of CORRECT answers is: 15
The total number of INCORRECT answers is: 5
Questions answered incorrectly are: 3 7 13 14 19

Submission:

You have to create your own files, and makefile that will compile the program successfully similar to the examples in the previous tasks. As the specific functions are not prescribed, the functions are not tested separately. The program is assessed as a unit.

7 Insurance Risk Calculations [75 marks]

You have been provided with a file called **claims.dat** which contains claims data for a company which insures motor cars. Each line contains the information for one claim and consists of the following fields:

- Policy number (integer)
- Claim type (string)
- Incident date (yyyy-mm-dd)
- Claim amount (real)
- Gender (string)
- Age (integer)
- Vehicle value (real)
- Year manufactured (integer)
- Colour (string)
- Immobilizer (bool)

Write a program to calculate the accident and theft risks for each of the policies. By obtaining the risk levels, the insurer can determine the premiums that can be paid by the clients. To determine the risk, you have to provide some points for each policy based on his/her accidental history, his personal details and the vehicle details. There are two types of risks that have to assessed for each policy holder, namely theft risk and accident risk. The points are integer values ranging from to 0 to 100 calculated based on the following rules.

The theft risk contributes a maximum of 40 points to the overall risk points. To calculate the theft risk points, the following rules are considered.

- If a vehicle do not have the security feature, an immobilizer, then the theft risk is increased by 10 points.
- Based on the condition that criminals are interested in stealing the latest models, a vehicle’s manufactured in 2013+, 2010 - 2012 and 2009 or earlier accumulates 10, 7 and 5 respectively.

- Vehicles in demand are expensive, therefore, a high vehicle value increases the theft risk points. The risk is determined as follows; cars costing at least R140000 accumulates a theft risk of 20 and those costing R100000 and above have a 10 risk otherwise the risk is 5.

To determine the accident risk, these constraints are to be followed.

- The number of accidents the vehicle holder was involved in increases the accident risk. If the policy holder was involved in single accident, twice, and at least three times then the risk points are respectively increased by 10, 20 and 30.
- Experience in driving is also used determining the risk value. Assuming all the policy holders started driving at the age of 20 years or less, policy holders with an age of at most 25 years old have a risk of 10 based on inexperience. Those who are 50 years old and above also have a risk of 10 based on the fact that their ability to concentrate and act fast is deteriorating. The other policy holders have an accident risk of 5.
- Vehicle repairing and compensation costs are determined by the vehicle purchase price. For this criteria, the risk breakdown is determined as follows; cars costing at least R140000 accumulates a risk of 20 and those costing R100000 and above have a 15 risk otherwise the risk is 10.

Here is the two dimension array structure of the insurance risk record.

PolicyID	accidentRisk	theftRisk	totalRisk	NumberOfAccidents

Table 2: Insurance Risk

Specification

Your program should define a two-dimensional integer array and populate it to hold the insurance risk records of the policies in the given data. You may assume that the number of policies does not exceed 2500. After populating the array, each policy should have one entry containing five integers representing the PolicyID, the accident risk, the theft risk, the total risk, and the number of accidents associated with the policy.

Implement the following functions that should be used in your program. Each one of these subtasks (units) will be marked independently by Fitchfork. Once all the functions have been completed and you have integrated them into a complete program, a final upload will be used to mark the integrated system.

`int findPolicy(int policyID, const int [][5] policyRiskInfo, int arrSize)`
Returns the index of the record containing the given policy ID number in the given 2D matrix. If the policy is not in the array, the function should return `arrSize`. The

first argument is the policy number, the second argument is the 2D array containing the risk information, and the last argument is the (row) size of the 2D array. You can apply any of the studied searching approach; binary search (since we read ordered files) or sequential searching, to find a particular policy record index.

`void showPolicy(int policyIndex, const int [][5] policyRiskInfo)`
Sends the content of the record at the given index in the given 2D matrix to standard output in a single descriptive line. The following is an example of the output the function should produce if called with `policyIndex` being 2.

```
2: Policy = 5, Accident = 25, Theft = 20, Total = 45, Accidents = 1
```

`void addPolicy(int policyID, int [][5] policyRiskInfo, int & arrSize)`
Populates the row at index `arrSize` in the given 2D array with a record containing the given policy number and zero values for the other fields. The value of `arrSize` is incremented. The first argument is the policy number, the second argument is the 2D array containing the risk information, and the last argument is the (row) size of the 2D array.

`int getAccidentRisk(int policyIndex, const int [][5] policyRiskInfo)`
Returns the value for accident risk that is stored record at the given index in the given 2D matrix.

`void setAccidentRisk(int policyIndex, int [][5] policyRiskInfo, int risk)`
Updates the value for accident risk in the record at the given index in the given 2D matrix to become the specified value that is passed in the `risk` parameter.

`int calculateAccidentRisk(int accidents, int age, double vValue)`
Use the accident risk constraints listed in the previous section to calculate the accident risk value based on the values that are passed as parameters to this function.

`int getTheftRisk(int policyIndex, const int [][5] policyRiskInfo)`
Returns the value for theft risk that is stored record at the given index in the given 2D matrix.

`void setTheftRisk(int policyIndex, int [][5] policyRiskInfo, int risk)`
Updates the value for theft risk in the record at the given index in the given 2D matrix to become the specified value that is passed in the `risk` parameter.

`int calculateTheftRisk(bool immobiliser, int year, double vValue)`
Use the theft risk constraints listed in the previous section to calculate the theft risk value based on the values that are passed as parameters to this function.

`int getTotalRisk(int policyIndex, const int [][5] policyRiskInfo)`
Returns the value for total risk that is stored record at the given index in the given 2D matrix.

`void updateTotalRisk(int policyIndex, int [][5] policyRiskInfo)`
Updates the value for total risk in the record at the given index in the given 2D matrix to become the sum of the current accident risk and the current theft risk in the record at the given index.

`int getAccidents(int policyIndex, const int [][5] policyRiskInfo)`
: Returns the value for number of accidents that is stored record at the given index in the given 2D matrix.

`void incAccidents(int policyIndex, int [][5] policyRiskInfo)`
: Increase the value for number of accidents in the record at the given index in the given 2D matrix with 1.

A Final Function

This function is meant to handle the intergration of the previously defined functions in order to fill-up the policy risk information 2D table.

`void fillUpPolicyRiskTable(int [][5] policyRiskInfo, int & arrSize)`
: This function must get the input values from **claims.dat** and correctly allocate values to the cells of the 2D array $policyRiskInfo$. After reading the policy data from each line in **claims.dat** a new policy is initialized if needed. The accident risk is calculated and the theft risk is calculated. All the records are then assigned with their respective values. If an update to accident risk is required, functions that handle such are called and the changes are effected.

User Interface

The integrated system should provide a simple text based menu driven interface giving the user access to the data stored in the 2D array. The following is the required menu:

```
Motor Vehicle Insurance: Client Risk
1. Find a policy and display its risks
2. List all policies and their risk values
3. Quit
Enter your choice (1-3):
------------------------------------------------------------
```

Output Example

The following is a sample test run of an integrated system that complies with the requirements for menu options 1 and 3. The implementation of menu option 2 is part of the challenges for this task.

```
Motor Vehicle Insurance: Client Risk
1. Find a policy and display its risks
2. List all policies and their risk values
3. Quit
Enter your choice (1-3): 1
------------------------------------------------------------
Enter the policy number: 5
```

```
3: Policy = 5, Accident = 35, Theft = 30, Total = 65, Accidents = 1
---------------------------------------------------------------
Motor Vehicle Insurance: Client Risk
1. Find a policy and display its risks
2. List all policies and their risk values
3. Quit
Enter your choice (1-3): 1
---------------------------------------------------------------
Enter the policy holder’s number: 3
Policy number 3 was NOT FOUND
---------------------------------------------------------------
Motor Vehicle Insurance: Client Risk
1. Find a policy and display its risks
2. List all policies and their risk values
3. Quit
Enter your choice (1-3): 3
---------------------------------------------------------------
Program Quits
---------------------------------------------------------------
```

Submissions

For this task, you have to create the following files:

makefile The instructions in this file should provide the correct detail to compile and/or run the integrated program.

main.cpp The driver of the integrated system that provides the required functionality:

- Declare the 2D array and open the file.
- Read the data from the file and populate the 2D array.
- Close the file.
- Provide the menu driven interface to access the data in the 2D array.

bigClaims.h The function prototypes for the prescribed functions as well as any additional functions you have implemented.

bigClaims.cpp The implementation of all the functions in your h-file.

Al these files (and nothing more) should be uploaded in each of the upload slots for fitchfork assessment. The following table describe the purpose of each of the provided slots for testing different units of your system:

Name	**Description**	**Method**
Policy	Testing the functions with names containing **Policy**	Using a driver that initiates the 2D array with appropriate test data
getters and setters	Testing all functions with names starting with **get** or **set** as well as **incAccidents** and **updateTotalRisk**	
calculators	Testing the functions with names starting with **calculate**	
System	Testing the integrated system in its entirety	Run the system with our own **claims.dat** file with well designed test data

Allocation of Marks

Name	Description	Marks
Policy	Testing the functions with names containing **Policy**	**addPolicy** [5 marks], **findPolicy** [3 marks], **showPolicy** [8 marks]
getters and setters	Testing all functions with names starting with **get** or **set** as well as **incAccidents** and **updateTotalRisk**	getters [2 marks], setters [3 marks], **incAccidents** [3 marks] & **updateTotalRisk**[2 marks]
calculators	Testing the functions with names starting with **calculate**	[10 marks] each
System	Testing the integrated system in its entirety	**fillUpPolicyRiskTable** [20 marks]
Total		**[75 marks]**

7.1 Challenges

- You have been introduced to std::vectors as an alternative to arrays, solve the aforementioned problem by using a vectors instead arrays. For more details on STL std::vectors see page 469 of your prescribed textbook.
- Implement the second option of the menu. As the number of items to be displayed is likely to exceed a typical screen, you have to show the output in chunks of 15 entries at a time. Leave an open line below 15 displayed entries followed by the following prompt:

  ```
  Enter Q to quit and go back to the menu or P to show the next page:
  ```

  The prompt should change to the following at the bottom of the last page:

  ```
  Enter Q to quit and go back to the menu or P to show the first page:
  ```

  In both cases the action when the appropriate letter was entered as upper case or lower case letter, should be executed. If the user enter anything else, the prompt should simply be re-displayed.
- Change the prototype and implementation of the `showPolicy` function so that it can be called with two parameters as in the non-challenge version, but can also be called using three parameters. If it is called with two parameters it's functionality should be as it was before this code change was implemented. This will ensure that existing code that relies on this function need not be changed as a result of this extension.

  The third parameter should be used to specify that the display should be in a format to allow a tabular display. When using the tabular display, the values in the individual entries should be listed without the descriptive text. This will enable the programmer to display the values of multiple recordes in a table with descriptive column titles.

References

[Kapoor and Kapoor(n.d.)] Kapoor VP and Kapoor PV (n.d.) Lo-Shu magic square, http://www.numeroworld.com/lo-shu-magic-square.asp. [Online; accessed 2015-03-26].

[Matt_Crypto(2013)] Matt_Crypto (2013) Caesar cipher with a left shift of 3, http://en.wikipedia.org/wiki/File:Caesar3.png. This work has been released into the public domain by its author.

[NRICH team(n.d.)] NRICH team (n.d.) An Introduction to Magic Squares, http://nrich.maths.org/2476. [Online; accessed 2015-03-26].

[Wikipedia(2015)] Wikipedia (2015) Caesar cipher — Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Caesar_cipher&oldid=649979932. [Online; accessed 25-March-2015].