

Assignment 1: Encryption

(25 points)

In this assignment we will encrypt a message (from the user) with a 'monoalphabetic substitution'. The procedure is quite simple: we create a substitution alphabet using a secret key and with this substitution alphabet (26 letters in a different order) we can change a clear text into unreadable text.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	standard alphabet
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	
s	e	c	r	t	k	y	a	b	d	f	g	h	i	j	l	m	n	o	p	q	u	v	w	x	z	subst. alphabet

Let's get started:

- Create a Console project with name '**Assignment1**', and give the solution the name 'Programming2-retake'.
(the other assignments will also be added as projects to this solution, see later)
- In the Start-method, ask the user to enter a message. Next, ask the user to enter a secret key that must be used to encrypt the message. Use lowercase letters.
- Create a method `string CreateSubstitutionAlphabet(string key, string standardAlphabet)` that creates a substitution alphabet using the key and the standard alphabet.
Follow the next procedure:
 - create a local variable 'alphabetTemp' and store the key and the standard alphabet in this local variable (after each other);
 - create a local variable 'substitutionAlphabet' and initialize it with an empty string;
 - iterate through all letters in alphabetTemp, and each time add the letter to substitutionAlphabet if it does not contain it yet;
 - return the substitution alphabet;
 → Call this CreateSubstitutionAlphabet method from the Start-method, using the following parameters: the entered secret key and the standard alphabet ("abc...xyz").
- Create a method `string ReplaceText(string input, string standardAlphabet, string substitutionAlphabet)` that encrypts the input text.
Follow the next procedure:
 - create a local variable 'output' and initialize it with an empty string;
 - iterate through all characters in (parameter) input, and for each character:
 - determine the position (of the character) in the standard alphabet using `IndexOf(...)`;
 - if the standard alphabet does not contain the character (position < 0) then add this character to output, otherwise add the character from the substitution alphabet to output (use the returned position as index in the substitution alphabet);
 - return the output;
 → Call this ReplaceText method from the Start-method, using the following parameters: the entered message, the standard alphabet, and the substitution alphabet.
→ In the Start-method, display the result. An example of the output is displayed below.

```

file:///C:/Users/Gerwin van Dijken/Documents/Visual...
Enter a message: this message must be encrypted...
Enter the secret key: secretkey
encrypted message: pabo htoosyt hqop et ticnxlptr...
  
```

Assignment 2: HighestDown / LowestUp matrix

(30 points)

In this assignment we will manipulate a 2-dimensional matrix.

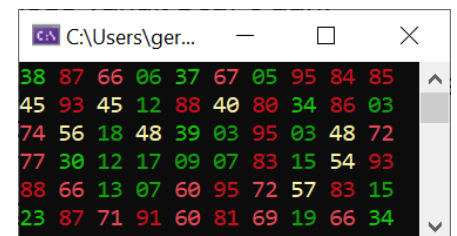
Let's get started:

0. Add to solution 'Programming2-retake' a Console project with name '**Assignment2**'.
1. In the Start method create a 2-dimensional `int`-array with the name "matrix"; give this matrix 6 rows and 10 columns (use 2 constants for this).

2. Implement a method `void FillMatrix(...)` that receives a 2-dimensional array as parameter. In this method fill the array with random numbers between 1 (inclusief) and 100 (exclusief). Use 2 constants for these min/max-values.
→ Call this `FillMatrix` method from the Start-method.

To get the same 'random' values as the screenshots, you can use Seed 1...

3. Implement a method `void DisplayMatrix(...)` that receives a 2-dim array as parameter. Use different colors for numbers between 1..19, 20..39, 40..59, 60..79, 80..99 (*hint: divide the matrix-value by 20, to get a color-index*). This method displays the matrix as can be seen in the screenshot to the right.
→ Call this `DisplayMatrix` method from the Start-method.



4. Implement a method `void HighestDown(int[,] matrix, int column)`. This method is storing the highest value in the given column (*parameter column*) at the bottom of the column.

Follow the next procedure:

- a) determine the row (index) of the highest value in the given column;
- b) swap this highest value with the value at the bottom of the column;

5. Implement a method `void ProcessMatrix(int[,] matrix)`. This method calls method `HighestDown` for every column. Make sure the code stills works if the number of columns changes!
→ From the Start-method: call `ProcessMatrix` and then `DisplayMatrix`. The output should look like the screenshot below in figure 2.1 (*to the left*).

6. Implement a method `void LowestUp(int[,] matrix, int column)`. This method is storing the lowest value in the given column at the top of the column. Use the same procedure as given for `HighestDown`.
→ Change method `ProcessMatrix` so for each column also method `LowestUp` is called. The output should look like the screenshot below in figure 2.2 (*to the right*).

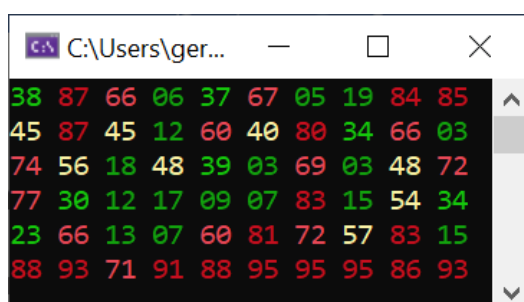


Figure 2.1: output item #5

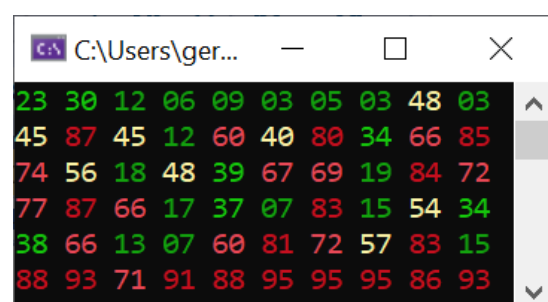


Figure 2.2: output item #6

Assignment 3: Election of House Representatives

(35 points)

Last month, the election of House Representatives in the Netherlands took place, with 37 political parties. In this election, the 150 seats ("zetels") are divided among the political parties, based on the number of votes received.

The votes are available for every city in the Netherlands, in file "TK2021-results.csv". Each line in this file contains the following fields: 1) name of the city, 2) total number of votes in this city, 3) the number of votes for the 1st party, 4) the number of votes for 2nd party, ..., 39) the number of votes for 37th party.
(source: <https://www.verkiezingsuitslagen.nl/verkiezingen/detail/TK20210317>)

Let's get started:

0. Add to solution 'Programming2-retake' a Console project with name '**Assignment3**'. Download file "TK2021-results.csv" (from Moodle) and save it in your project directory.

1. Create a `class CityResult` with the fields `CityName (string)`, `TotalVotes (int)` and `PartyVotes (int[])`; store this class in a separate file with the name "CityResult.cs".

Create a `class PoliticalPartyResult` with the fields `Name (string)`, `TotalVotes (int)` and `NrOfSeats (int)`; store this class in a separate file with the name "PoliticalPartyResult.cs".
(this class is not used until item #5, see next page)

2. Create a method with signature:

`List<string> ReadPoliticalParties(string filename)`

In this method all political parties are read (from file <filename>) and added to a list. You only need to read the first line, since this line contains all party names. The fields are separated by a semicolon, so use `Line.Split(';')`. Skip the first two fields, the remaining fields contain the party names.

Make sure the program doesn't crash if the file does not exist!

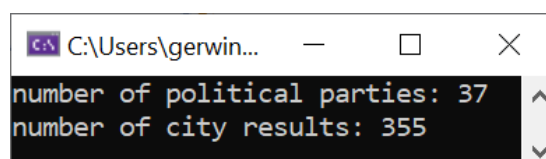
3. Create a method with signature:

`List<CityResult> ReadCityResults(string filename, int nrOfPoliticalParties)`

In this method the vote-results of all cities are read (from file <filename>) and added to a list. The fields are separated by a semicolon, so use `Line.Split(';')`. Skip the first line (header). For each `CityResult` object, create array `PartyVotes` (use parameter `nrOfPoliticalParties` for the length of this array); use a loop to fill this array. When all lines have been processed, return the list with city results.

Make sure the program doesn't crash if the file does not exist!

4. Call methods `ReadPoliticalParties` and `ReadCityResults` from the `Start` method, use file "TK2021-results.csv" for both methods. Stop the program when there are no items in one of the (received) lists. You may want to print the number of items in the lists, as a check (or use the Debugger, for checking these numbers).



```
C:\Users\gerwin...
number of political parties: 37
number of city results: 355
```

5. Create a method with signature:

```
List<PoliticalPartyResult> GetPartyResults(List<string> politicalParties,
                                           List<CityResult> cityResults)
```

This method determines for each political party the total number of votes, so the sum of votes in all cities.

Follow the next procedure:

- create a list for storing `PoliticalPartyResult` objects, and for each political party add a `PoliticalPartyResult` object to this list.
- process all city results: for each city, add the votes for all parties in the corresponding party-results (so, iterate through array `PartyVotes`);
- return the list;

6. Create a method with signature:

```
void DisplayPartyResults(List<PoliticalPartyResult> partyResults)
```

This method displays for all political parties the total number of votes, see figure 3.1 below.

→ Call method `DisplayPartyResults` from the `Start` method;

7. Create a method with signature:

```
void CalculatePartySeats(List<PoliticalPartyResult> partyResults)
```

This method determines for all parties the number of (House Representatives) seats.

Follow the next procedure:

- determine the total number of votes, by adding the votes of all parties;
- divide the total number of votes by the number of seats (150), the "electoral quota" (`int`);
- now give each party the number of "full seats" → the total number of party votes divided by the electoral quota;

→ Call method `CalculatePartySeats` from the `Start` method, and update method `DisplayPartyResults`; an example of the output is given in figure 3.2 below;

8. Create a method with signature: **Assignment 8 is a bonus one (5 extra points), max grade remains 100**

```
void DivideRemainderSeats(List<PoliticalPartyResult> partyResults,
                          int nrOfRemainderSeats)
```

In the election, there are always some "remainder seats", the total number of seats (150) minus the number of "full seats" (see item #7). Use the following procedure ("highest averages method") for dividing the remainder seats:

- create a `double`-array "averageVotes" and store for each party the next value in this array: the total number of party-votes divided by (the number of received seats + 1);
- now process all remainder seats one by one (in a loop): determine the party with the highest value in the array (the party with the "highest average"), give this party an extra seat and update the average (in array `averageVotes`) with the formula given in a;

→ Call method `DivideRemainderSeats` in method `CalculatePartySeats`; an example of the output is given in figure 3.3 below.

Party	Votes
VVD	2279126
D66	1565862
PVV	1124482
CDA	990601
SP	623371
PvdA	597192
GROENLINKS	537308
FvD	523083

Figure 3.1: output item #6

Party	Votes	Seats
VVD	2279126	32
D66	1565862	22
PVV	1124482	16
CDA	990601	14
SP	623371	8
PvdA	597192	8
GROENLINKS	537308	7
FvD	523083	7

Figure 3.2: output item #7

Party	Votes	Total Seats
VVD	2279126	34
D66	1565862	24
PVV	1124482	17
CDA	990601	15
SP	623371	9
PvdA	597192	9
GROENLINKS	537308	8
FvD	523083	8

Figure 3.3: output item #8