Assignment 1: ISBN

(25 points)

In this assignment we will check ISBN codes. An ISBN code is used to uniquely identify a book. There are 2 types of ISBN codes: a 10-digit ISBN format and a 13-digit ISBN format (since 1 January 2007). In this assignment we will only check the 13-digit format.

ISBN 979-1-23456-789-6

Let's get started:

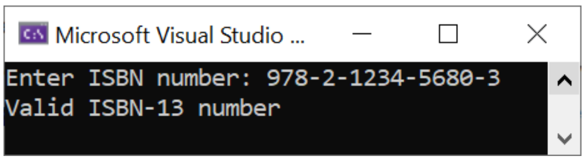
0. Create a Console project with name '**Assignment1**', and give the solution the name 'Programming2-Exam'.
   (*the other assignments will also be added as projects to this solution, see later)*

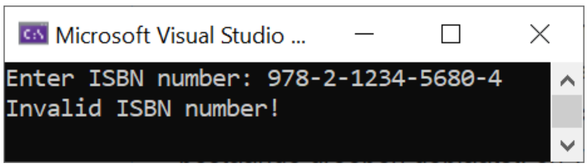
1. Create an enumeration `ISBNValidation` with options `InvalidISBN` and `ValidISBN13`. Store the enumeration in a separate file.

2. Create a method with signature: `ISBNValidation` **`ValidateISBN`**`(string isbn)`.
   This method returns either `ValidISBN13` or `InvalidISBN` *(and maybe in the future also `ValidISBN10`...)*. If the isbn contains at least 13 characters, use method `IsValidISBN13` (see next item) to see if it's a valid isbn.

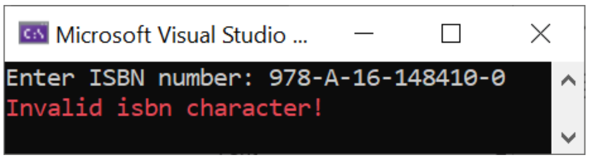
3. Create a method with signature: `bool IsValidISBN13(string isbn)`.
   This method returns `true` if the given isbn is a 13-digit ISBN format, `false` otherwise.
   Follow the next procedure:
   a. remove all hyphens ('-') from the given isbn by using: `isbn = isbn.Replace("-", "");`
   b. if the *(stripped)* isbn does not have length 13, return `false`;
   c. determine the sum of all 13 digits (with a loop), where each digit at an even position (2nd, 4th, …) is multiplied by 3. To convert a char to a digit (0..9), you can use `isbn[i] - '0'`.
   An example: 978-2-1234-5680-3 → sum = 9 + 7*3 + 8 + 2*3 + 1 + 2*3 + 3 + 4*3 + 5 + 6*3 + 8 + 0*3 + 3 = 100;
   d. if the sum is a multiple of 10, return `true`, else return `false`;

4. In the `Start` method, ask the user to enter an ISBN number. Call method `ValidateISBN` and use a switch-statement to process the return value (`ValidISBN13` or `InvalidISBN`).


```
Enter ISBN number: 978-2-1234-5680-3
Valid ISBN-13 number
```


```
Enter ISBN number: 978-2-1234-5680-4
Invalid ISBN number!
```

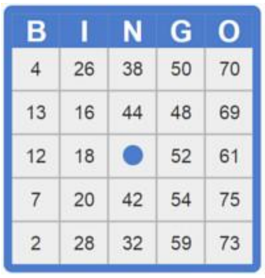
5. Modify method `IsValidISBN13`: throw an exception (with message "Invalid isbn character!") if the isbn contains a non-digit (not 0..9). Catch the exception in the `Start` method, and display the message of this exception.


```
Enter ISBN number: 978-A-16-148410-0
Invalid isbn character!
```

Assignment 2: Bingo

(25 points)

In this assignment we will fill a bingo card with random numbers. A bingo card can be implemented as a matrix with 5 rows and 5 columns, containing numbers between 1 and 75 (see example to the right). Each number can occur only once. The possible column numbers are: in the 1st column ('B'): 1..15, in the 2nd column ('I'): 16..30, in the 3rd column ('N'): 31..45, in the 4th column ('G'): 46..60 and in the 5th column ('O'): 61..75. The center cell is always empty.



B	I	N	G	O
4	26	38	50	70
13	16	44	48	69
12	18	●	52	61
7	20	42	54	75
2	28	32	59	73

Aan de slag:

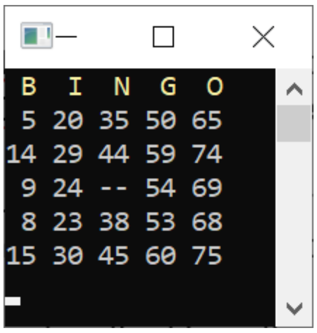
0. Add to solution 'Programming2-Exam' a Console project with name '**Assignment2**'.

1. In the `Start` method, create a 2-dimensional `int`-array with the name "bingoCard"; give this matrix 5 rows and 5 columns (use 2 constants for this).

2. Implement a method with signature: `void FillBingoColumn(int[,] bingoCard, int column, int minNumber, int maxNumber)`.
   In this method one complete column (parameter <column>) of the bingo card is filled with random numbers between <minNumber> (inclusive) and <maxNumber> (inclusive). Make sure a number occurs only once.

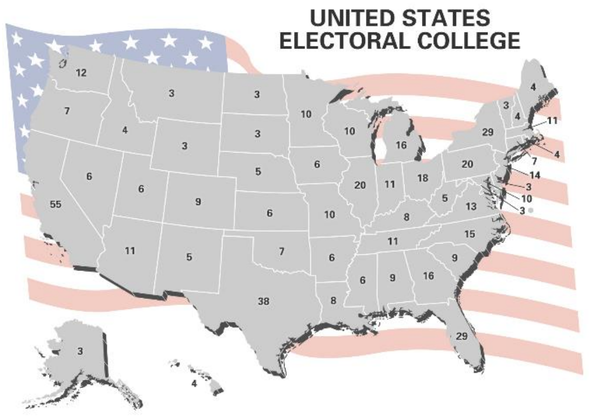
3. Implement a method **`FillBingoCard`** that receives a 2-dimensional array as parameter. This method calls method `FillBingoColumn` for each column in the bingo card (use a loop), in order to fill the complete bingo card with random numbers. For the 1st column use 1..15 as min..max, for the 2nd column use 16..30, etcetera.
   Make sure that after filling the complete bingo card, the center cell is cleared (by storing the number 0).
   → Call this `FillBingoCard` method from the `Start` method.

4. Implement a method **`DisplayBingoCard`** that receives a 2-dimensional array as parameter. This method displays the array as shown in the screenshot to the right. Make sure this display-method can also display bingo cards with more than 5 rows/columns. For invalid numbers (smaller/equal 0) display a '--'.
   → Call this `DisplayBingoCard` method from the `Start` method.



```
 B  I  N  G  O
 5 20 35 50 65
14 29 44 59 74
 9 24 -- 54 69
 8 23 38 53 68
15 30 45 60 75
```

Assignment 3: US presidential election (40 points)



The US presidential election took place on November 3rd. A candidate must receive at least 270 electoral votes (electors) to become the next president. There are 538 electors in total, these are allocated among the states *(see picture to the right)*. The candidate that receives the most votes within a state, wins the electors of that state.

The state results of the US election of 2020 must be read from file "2020-votes.csv"; on each line the following information is stored: 1) state name, 2) candidate name, 3) party name, and 4) number of votes.
The number of electors for each state must be read from file “2020-electors.csv”; on each line: 1) state name, and 2) number of electors. *(data downloaded from https://electionlab.mit.edu/data)*

Let’s get started:

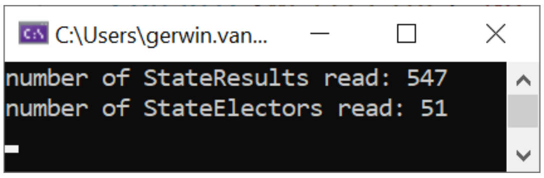
0. Add to solution ‘Programming2-Exam’ a <u>Console</u> project with name ‘**Assignment3**’. Download files "2020-votes.csv" and “2020-electors.csv” (from Moodle) and save it in your projectdirectory.

1. Create a `class StateResult` with fields StateName (`string`), CandidateName (`string`), PartyName (`string`), and CandidateVotes (`int`). Store this class in a separate code file with name "StateResult.cs".
   Create a `class StateElectors` with fields StateName (`string`), and ElectorsCount (`int`). Store this class in a separate code file with name "StateElectors.cs".

2. Create a method: `List<StateResult> ReadStateResults(string filename)`. In this method all state results are read (from the given file) and added to a list. The fields in the file are separated by a semicolon, so use: `line.Split(';')`. When all lines have been read/processed, return the list with results. Make sure the program does not crash if the file does not exist!

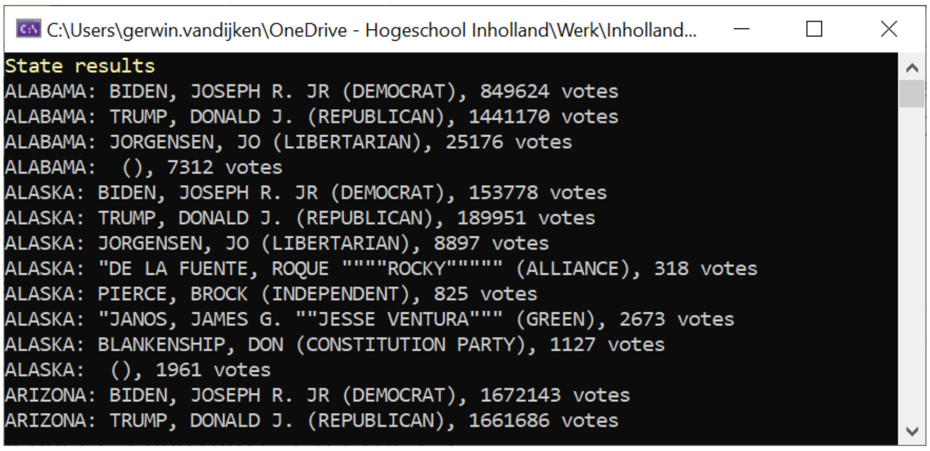
3. Create a method: `List<StateElectors> ReadStateElectors(string filename)`. In this method all state electors are read (from the given file) and added to a list. The fields in the file are separated by a semicolon, so use: `line.Split(';')`. When all lines have been read/processed, return the list with results. Make sure the program does not crash if the file does not exist!

4. Call method `ReadStateResults` and `ReadStateElectors` from the `Start` method, passing the corresponding filename ("2020-votes.csv" or “2020-electors.csv”). Stop the program when there are no entries in a list. You may want to print the number of items read, as a check (or use the Debugger, for checking these numbers).



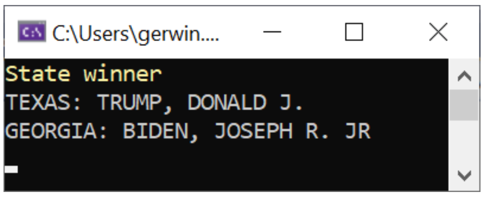
```
number of StateResults read: 547
number of StateElectors read: 51
```

5. Create a method with signature: `void DisplayStateResults(List<StateResult> results)`. This method displays all state results, see example below.



```
State results
ALABAMA: BIDEN, JOSEPH R. JR (DEMOCRAT), 849624 votes
ALABAMA: TRUMP, DONALD J. (REPUBLICAN), 1441170 votes
ALABAMA: JORGENSEN, JO (LIBERTARIAN), 25176 votes
ALABAMA:  (), 7312 votes
ALASKA: BIDEN, JOSEPH R. JR (DEMOCRAT), 153778 votes
ALASKA: TRUMP, DONALD J. (REPUBLICAN), 189951 votes
ALASKA: JORGENSEN, JO (LIBERTARIAN), 8897 votes
ALASKA: "DE LA FUENTE, ROQUE """"ROCKY""""" (ALLIANCE), 318 votes
ALASKA: PIERCE, BROCK (INDEPENDENT), 825 votes
ALASKA: "JANOS, JAMES G. ""JESSE VENTURA""" (GREEN), 2673 votes
ALASKA: BLANKENSHIP, DON (CONSTITUTION PARTY), 1127 votes
ALASKA:  (), 1961 votes
ARIZONA: BIDEN, JOSEPH R. JR (DEMOCRAT), 1672143 votes
ARIZONA: TRUMP, DONALD J. (REPUBLICAN), 1661686 votes
```

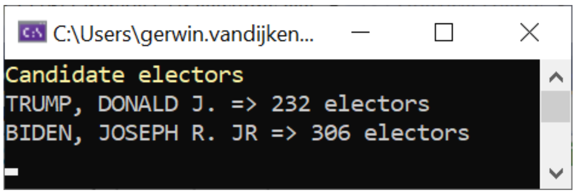
6. Create a method with signature:
   `string WinnerOfState(List<StateResult> stateResults, string stateName)`.
   This method returns the name of the candidate with the most votes in the given state.
   → You may want to call this method from the Start method, to check the method (see some examples below, for states "TEXAS" and "GEORGIA").



```
State winner
TEXAS: TRUMP, DONALD J.
GEORGIA: BIDEN, JOSEPH R. JR
```

7. Create a method with signature:
   ```
   Dictionary<string, int> GetCandidateElectors(List<StateResult> stateResults,
                                                List<StateElectors> stateElectors).
   ```
   This method returns a dictionary containing for each unique candidate (key:`string`) the total number of elector votes he/she won (value:`int`). Process all items in list stateElectors (51x), and for each item: get the winner of the state (using method `WinnerOfState`), and update the dictionary using the winner and number of state electors *(if the dictionary does not contain the candidate (winner) yet, then add a new entry in the dictionary, else update it)*. Finally, return the dictionary.

8. Create a method with signature:
   `void DisplayCandidateElectors(Dictionary<string, int> candidateElectors)`
   This method displays all entries of the given dictionary.
   → Call methods `GetCandidateElectors` and `DisplayCandidateElectors` from the `Start` method (see output below).



```
Candidate electors
TRUMP, DONALD J. => 232 electors
BIDEN, JOSEPH R. JR => 306 electors
```