# Exercises: Methods

Problems for exercises and homework for the [“Programming Fundamentals Extended” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

You can check your solutions here: <https://judge.softuni.bg/Contests/418>.

## Hello, Name!

Write a **method** that receives a name as **parameter** and prints on the console. “Hello, <name>!”

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter | Hello, Peter! |

## Min Method

Create a GetMin(int a, int b) method, that returns the **smaller** of two numbers. Write a program that reads three numbers from the console and prints the **smallest** of them. Use the GetMin(…) method you just created.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1  2  3 | 1 |  | -100  -101  -102 | -102 |

## String Repeater

Write a **RepeatString(str, count)** method that receives a **string** and an **integer** **N** as **parameters** and returns the string, **repeated** **N** times. After that, print the result on the console.

|  |
| --- |
| static string RepeatString(string str, int count)  {  string repeatedString = string.Empty;    for (int i = 0; i < count; i++)  {  // TODO  }  return repeatedString;  } |

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| str  2 | strstr | roki  6 | rokirokirokirokirokiroki |

## Nth Digit

Write a **method** FindNthDigit(number, index) that receives a **number** and an **index N** as **parameters** and returns the **Nth digit** of the number (counted from **right to left**, starting from **1**). After that, **print** the result on the console.

|  |
| --- |
| static int FindNthDigit(long number, int index)  {  …  } |

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 83746  2 | 4 | 93847837  6 | 8 | 2435  4 | 2 |

### Hints

* Use a while loop to perform the algorithm until the given number becomes **0**.
* On every iteration of the while loop, check if the **current index** of the number equals the **index** you are looking for. If it is, return the number at that index (number % 10). If it’s not the correct index, **remove** the last digit from the number (number / 10).
* Keep track of which **digit** you are checking by **index** (**right to left**, starting from **1**). When you **find** the digit, **return** that **index**.

## Integer to Base

Write a **method** **IntegerToBase(number, toBase)** that receives a **number** and a **base** as **parameters** and returns the number, **converted** to that **number base.** After that, **print** the result on the console.

The **number** will **always** be in **base 10** and the **base** parameter will **always** be **between 2 and 10**.

|  |
| --- |
| static string IntegerToBase(long number, int toBase)  {  …  } |

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3  2 | 11 | 4  4 | 10 | 9  7 | 12 |

### Hints

* Declare a variable where the **result** will be stored.
* Perform the calculations, needed to convert the number:
  + Take the **remainder** of the **number**, divided by the **base.**
  + Add the **remainder** to the **front** of the result string.
  + Divide the **number** by the **base**
  + The above algorithm should run **until** the number is **0**

## Notifications

Write 2 methods:

|  |
| --- |
| static string ShowSuccess(string operation, string message)  {  …  } |

which prints a message in the following format:

|  |
| --- |
| Successfully executed {operation}.  ==============================  Message: {message}. |

and

|  |
| --- |
| static string ShowError(string operation, int code)  {  …  } |

which prints a message in the following format:

|  |
| --- |
| Error: Failed to execute {operation}.  ==============================  Error Code: {code}.  Reason: {reason}. |

The **reason** is determined by the **code**:

* If the code is **positive**, the reason is "Invalid Client Data"
* If the code is **negative**, the reason is "Internal System Failure"

Then, create a program which accepts an integer **N**.

On the next **N** lines, the user receives a **result** – "success" or "error". If the current operation is success, it should prompt the user for an **operation** and a **message**. If the operation is error, it should prompt the user for an operation and a code.

If neither "success", nor "error" is given, **nothing** should be printed and the program should **continue**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  success  User registration  User registered successfully  error  connecting to server  -403 | Successfully executed User registration.  ==============================  Message: User registered successfully.  Error: Failed to execute connecting to server.  ==============================  Error Code: -403.  Reason: Internal System Failure. |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  success  Database query  Updated 2 rows  warning  error  credit card purchase  2 | Successfully executed Database query.  ==============================  Message: Updated 2 rows.  Error: Failed to execute credit card purchase.  ==============================  Error Code: 2.  Reason: Invalid Client Data. |

## \* Numbers to Words

Write a method Letterize(number) which accepts a number and prints it as **words**, according to these conditions:

* If the number is more than 999 you should print – "too large"
* If the number is less than -999 you should print – "too small"
* If the number is negative, you should print "minus" before it.
* If the number does **not** have 3 digits, **do not** print it

The program should accept an integer **N.** On the next **N lines,** you should accept **numbers** and print them as **words**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3  999  -420  1020 | nine-hundred and ninety nine  minus four-hundred and twenty  too large | 2  15  350 | three-hundred and fifty |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 4  311  418  519  -9945 | three-hundred and eleven  four-hundred and eighteen  five-hundred and nineteen  too small | 2  500  123 | five-hundred  one-hundred and twenty three |

### Hints

* The first set of special cases comes when a number’s right on the **hundreds** (i.e. 100, 200, 300, etc.). Print them like this: “one-hundred”, “two-hundred”, “three-hundred”.
* The second set of special cases comes when a number’s last 2 digits are **less than 10** (i.e. 101, 305, 609, etc.). Print them like this: “one-hundred and one”, “three-hundred and five”, “six-hundred and nine”
* The third set of special cases comes when a number is in the **teens** (i.e. **111, 814, 919**). Print them like this: “one-hundred and eleven”, “eight-hundred and fourteen”, “nine-hundred and nineteen”

## \* String Encryption

Write a method Encrypt(char letter) which encrypts the letter in the following way:

* Take the first and the last digit of its **ASCII** code and append them together in a string.
* Append at the **start** of the resulting string the **character corresponding to**:
  + the **ASCII code** of the letter **+** the **last digit** of the **ASCII code** of the letter
* Then append at the **end** of the resulting string the **character corresponding to**:
  + the **ASCII code** of the letter **-** the **first digit** of the **ASCII code** of the letter
* The method should return the encrypted string.

Example

* j -> p16i
  + j’s ASCII code is **106 🡺** First digit – **1**, last digit – **6**
  + Append the first and last digit 🡺 **16**
  + Append the character, resulting from the ASCII code + last digit to the **start** 🡺 106 + 6 🡺 112 🡺 p
  + Append the character, resulting from the ASCII code - first digit to the **end** 🡺 106 - 1 🡺 105 🡺 i
  + End result 🡺 p16i

Assume that the input for this method will **always** be valid.

The main program should read an input from the user - an integer **N**.

On the next **N lines,** you will receive **characters**. **Encrypt** them and **append** them to an **encrypted string**.

In the end, what you should have is a long **encrypted string** of characters like this:

* S, o, f, t, U, n, i 🡺 V83Kp11nh12ez16sZ85Mn10mn15h

Your task is to **print** the **encrypted string.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 7  S  o  f  t  U  n  i | V83Kp11nh12ez16sZ85Mn10mn15h |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 7  B  i  r  a  H  a  x | J72Ah97Xl99Zr17jH66<n15hv14qh97X |

### Hints

* Start with an empty **result** string – use string.Empty.
* Loop **n** times and each time **append** the **encrypted character** to the result string.
* To get the **first** and **last** digits of the **ASCII** code, use the same algorithm as the one you’d use to solve the **Integer to Base** problem.