# Exercises: C# Basics – More Exercises

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

## X

Write a program, which **prints** an **X figure** with height **n**.

**N** will be an **odd number** in the range **[3…99]**.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3 | x x  x  x x | 5 | x x  x x  x  x x  x x | 11 | x x  x x  x x  x x  x x  x  x x  x x  x x  x x  x x |

## Vapor Store

After the previous problem, you feel like taking a break, so you go on the **Vapor Store** to buy some video games. Write a program, whichhelps you buy the games. The **valid games** are the following games in this table:

|  |  |
| --- | --- |
| **Name** | **Price** |
| OutFall 4 | $39.99 |
| CS: OG | $15.99 |
| Zplinter Zell | $19.99 |
| Honored 2 | $59.99 |
| RoverWatch | $29.99 |
| RoverWatch Origins Edition | $39.99 |

On the first line, you will receive your **current balance** – a **floating-point** number in the range **[0.00…5000.00]**.

Until you receive the command “Game Time”, you have to keep **buying games**. When a **game** is **bought**, the user’s **balance** decreases by the **price** of the game.

Additionally, the program should obey the following conditions:

* If a game the user is trying to buy is **not present** in the table above, print “Not Found” and **read the next line**.
* If at any point, the user has **$0** left, print “Out of money!” and **end the program**.
* Alternatively, if the user is trying to buy a game which they **can’t afford**, print “Too Expensive” and **read the next line**.

When you receive “Game Time”, **print** the user’s **remaining money** and **total spent on games**, **rounded** to the **2nd decimal place**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 120  RoverWatch  Honored 2  Game Time | Bought RoverWatch  Bought Honored 2  Total spent: $89.98. Remaining: $30.02 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 19.99  Reimen origin  RoverWatch  Zplinter Zell  Game Time | Not Found  Too Expensive  Bought Zplinter Zell  Out of money! |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 79.99  OutFall 4  RoverWatch Origins Edition  Game Time | Bought OutFall 4  Bought RoverWatch Origins Edition  Total spent: $79.98. Remaining: $0.01 |

## Megapixels

Write a program, which, given an **image resolution** (**width** and **height**), calculates its **megapixels**. **Megapixels** (short for **millions of pixels**) are calculated by **counting** all the **image** **pixels**, then **dividing** the result by **1000000**.

The megapixels must **always** be rounded to the **first digit** after the **decimal point** (i.e. 0.786 MP 🡺 0.8MP).

### Input

* **First Line** – the **width** of the image – integer in range **[1…20000]**
* **Second Line** – the **height** of the image – integer in range **[1…20000]**

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1024  768 | 1024x768 => 0.8MP | 1920  1080 | 1920x1080 => 2.1MP | 5344  3006 | 5344x3006 => 16.1MP |

### Hints

* To round a number, you can use the [Math.Round()](https://msdn.microsoft.com/en-us/library/75ks3aby(v=vs.110).aspx) method.

## Photo Gallery

Write a program, which receives **image metadata** as input and prints information about the image, such as its **filename**, **date taken**, **size**, **resolution** and **aspect ratio**. Also, calculate the **orientation** of the image. The **3** orientations are: **portrait**, **landscape** and **square**.

### Input

* **First line** – the photo’s **number** – an **integer** in the range **[0…9999]**
* **Second, third, fourth line** – the **day**, **month** and **year** the photo was taken – **integers** forming **valid** dates in the range **[01/01/1990…31/12/2020]**
* **Fifth, sixth line** – the **hours** and **minutes** the photo was taken – **integers** in the range **[0…23]**
* **Seventh line** – the photo’s **size** in **bytes** – **integer** in the range **[0…999000000]**
* **Eighth, ninth line** – the photo’s **resolution** (**width** and **height**)in **pixels** – **integers** in the range **[1…10000]**

### Output

* The **name** should be printed in the format “DSC\_xxxx.jpg”.
* The **date and time taken** should be printed in the format “dd/mm/yyyy hh:mm”.
* The **size** should be printed in standard **human-readable** format (i.e. 950 bytes = 950B, 500000 bytes = 500KB, 1500000 bytes = 1.5MB).
* The **resolution** should be printed in the following format: “{width}x{height}”.
* The **orientation** can be one of three valid values: **portrait**, **landscape** and **square**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 35  25  12  2003  12  3  1500000  5334  3006 | Name: DSC\_0035.jpg  Date Taken: 25/12/2003 12:03  Size: 1.5MB  Resolution: 5334x3006 (landscape) | 5**33**  **20**  **3**  **1993**  **11**  **33**  **350000**  **768**  **1024** | Name: DSC\_0533.jpg  Date Taken: 20/03/1993 11:33  Size: 350KB  Resolution: 768x1024 (portrait) |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6552  12  11  2012  15  33  10  1000  1000 | Name: DSC\_6552.jpg  Date Taken: 12/11/2012 15:33  Size: 850B  Resolution: 1000x1000 (square) |
|  |  |

## BPM Counter

Write a program, which receives **BPM** (beats per minute) and **number of beats** from the console and calculates how many **bars** (1 bar == 4 beats) the beats equal to, then calculates the **length** of the sequence in **minutes** and **seconds**.

The bars must **always** be rounded to the **first digit** after the **decimal point** (i.e. 1.75 bars 🡺 1.8 bars).

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 60  60 | 15 bars - 1m 0s | 128  85 | 21.2 bars - 0m 39s | 522  80 | 20 bars - 0m 9s |

## DNA Sequences

You are a molecular biologist, who’s on the verge of figuring out gene manipulation. But first you need to see what DNA sequences you’re working with, so you decide to write a program to do it for you.

Write a program, which prints all the possible **nucleic acid sequences** (A, C, G and T), in the **range** [**AAA…TTT**]. **Each** nucleic acid sequence is exactly **3 nucleotides (letters) long**. Print a **new line** every **4** sequences.

Each nucleotide has a corresponding numeric value – A 🡺 1, C 🡺 2, G 🡺 3, T 🡺 4.

For every sequence, take the **sum** of its elements (e.g. **ACAC** 🡺 1 + 2 + 1 + 2 = 6) and if it’s **equal to or larger than** the **match sum**, print the sequence with an “O” before and after it, otherwise print “X” before and after it.

### Examples

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** | **Comments** |
| 5 | XAAAX XAACX OAAGO OAATO  XACAX OACCO OACGO OACTO  OAGAO OAGCO OAGGO OAGTO  OATAO OATCO OATGO OATTO  XCAAX OCACO OCAGO OCATO  OCCAO OCCCO OCCGO OCCTO  OCGAO OCGCO OCGGO OCGTO  OCTAO OCTCO OCTGO OCTTO  OGAAO OGACO OGAGO OGATO  OGCAO OGCCO OGCGO OGCTO  OGGAO OGGCO OGGGO OGGTO  OGTAO OGTCO OGTGO OGTTO  OTAAO OTACO OTAGO OTATO  OTCAO OTCCO OTCGO OTCTO  OTGAO OTGCO OTGGO OTGTO  OTTAO OTTCO OTTGO OTTTO | 11 | XAAAX XAACX XAAGX XAATX  XACAX XACCX XACGX XACTX  XAGAX XAGCX XAGGX XAGTX  XATAX XATCX XATGX XATTX  XCAAX XCACX XCAGX XCATX  XCCAX XCCCX XCCGX XCCTX  XCGAX XCGCX XCGGX XCGTX  XCTAX XCTCX XCTGX XCTTX  XGAAX XGACX XGAGX XGATX  XGCAX XGCCX XGCGX XGCTX  XGGAX XGGCX XGGGX XGGTX  XGTAX XGTCX XGTGX OGTTO  XTAAX XTACX XTAGX XTATX  XTCAX XTCCX XTCGX XTCTX  XTGAX XTGCX XTGGX OTGTO  XTTAX XTTCX OTTGO OTTTO | Combinations, where “sum >= 11”:  GTT 🡺 3+4+4 🡺 11  TGT 🡺 4+3+4 🡺 11  TTG 🡺 4+4+3 🡺 11  TTT 🡺 4+4+4 🡺 12 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 10 | XAAAX XAACX XAAGX XAATX  XACAX XACCX XACGX XACTX  XAGAX XAGCX XAGGX XAGTX  XATAX XATCX XATGX XATTX  XCAAX XCACX XCAGX XCATX  XCCAX XCCCX XCCGX XCCTX  XCGAX XCGCX XCGGX XCGTX  XCTAX XCTCX XCTGX OCTTO  XGAAX XGACX XGAGX XGATX  XGCAX XGCCX XGCGX XGCTX  XGGAX XGGCX XGGGX OGGTO  XGTAX XGTCX OGTGO OGTTO  XTAAX XTACX XTAGX XTATX  XTCAX XTCCX XTCGX OTCTO  XTGAX XTGCX OTGGO OTGTO  XTTAX OTTCO OTTGO OTTTO | Combinations, where “sum >= 10”:  CTT 🡺 2+4+4 🡺 10  GGT 🡺 3+3+4 🡺 10  GTG 🡺 3+4+3 🡺 10  GTT 🡺 3+4+4 🡺 11  TCT 🡺 4+2+4 🡺 10  TGG 🡺 4+3+3 🡺 10  TGT 🡺 4+3+4 🡺 11  TTC 🡺 4+4+2 🡺 10  TTG 🡺 4+4+3 🡺 11  TTT 🡺 4+4+4 🡺 12 |

## Training Hall Equipment

As the new intern in SoftUni, you’re tasked with **equipping** the **new training halls** with all the **necessary items** to lead quality technical trainings. You’ll be given a **budget** and a **list of items** to buy. The other intern will be tasked with plugging in everything and hopefully not getting anyone electrocuted in the process…

### Input

* On the first line, you will receive your **budget** – a floating-point value in the range **[0…1000000]**
* On the second line, you will receive the **number of items** you need to buy – an integer in the range **[0…10]**
* On the next **count\*3** lines, you will receive the **item data** as such:
  1. The **item** **name** – **string**
  2. The **item price** – **floating-point** value in the range **[0.50…1000.00]**
  3. The **item count** – **integer** in the range **[0…1000]**

### Output

Every time an item is **added** to the cart, print “Adding {count} {item} to cart.” on the console. Make sure to **pluralize** item names (if the **item** **count** **isn’t 1**, add an **S** at the end of the item name). After all of the items have been added to the **cart**, you need to calculate the **subtotal** of the items and check if the **budget** will be **enough**.

* If it’s enough, print “Money left: ${moneyLeft}”, **formatted** to the **2nd decimal point**.
* Otherwise, print “Not enough. We need ${moneyNeeded} more.”, **formatted** to the **2nd decimal point**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 20000  4  projector  299.99  2  hdmi cable  4.99  1  chair  19.99  180  desk  199.99  60 | Adding 2 projectors to cart.  Adding 1 hdmi cable to cart.  Adding 180 chairs to cart.  Adding 60 desks to cart.  Subtotal: $16202.57  Money left: $3797.43 | 700  3  projector  399.99  1  hdmi cable  6.99  3  chair  2.99  80  desk  99.99  25 | Adding 1 projector to cart.  Adding 3 hdmi cables to cart.  Adding 80 chairs to cart.  Subtotal: $660.16  Money left: $39.84 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2000  4  whiteboard  150  1  marker  6.99  10  chalk  0.50  20  beanbag chair  119.99  15 | Adding 1 whiteboard to cart.  Adding 10 markers to cart.  Adding 20 chalks to cart.  Adding 15 beanbag chairs to cart.  Subtotal: $2029.75  Not enough. We need $29.75 more. |

## \* SMS Typing

Write a program, which emulates **typing an SMS**, following this guide:

|  |  |  |
| --- | --- | --- |
| **1** | **2**  abc | **3**  def |
| **4**  ghi | **5**  jkl | **6**  mno |
| **7**  pqrs | **8**  tuv | **9**  wxyz |
|  | **0**  space |  |

Following the guide, **2** becomes “**a**”, **22** becomes “**b**” and so on.

### Input

* On the first line, you will receive **n** - the **number of characters** – **integer** in the range **[1…30]**
* On the next n lines, you will receive integers, representing the **text message characters**.

### Output

Print all the characters together, forming a **text message string**.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 5  44  33  555  555  666 | hello | 9  44  33  999  0  8  44  33  777  33 | hey there | 7  6  33  33  8  0  6  33 | meet me |

### Hints

* A naïve approach would be to just put all the possible combinations of digits in a giant switch statement.
* A cleverer approach would be to come up with a **mathematical formula**, which **converts** a **number** to its **alphabet** representation:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Digit** | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **Index** | 0 1 2 | 3 4 5 | 6 7 8 | 9 11 12 | 13 14 15 | 16 17 18 19 | 20 21 22 | 23 24 25 26 |
| **Letter** | 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 |

* Let’s take the number **222** (**c**) for example. Our algorithm would look like this:
  + Find the **number of digits** the number has “e.g. **222** 🡺 **3 digits**”
  + Find the **main digit** of the number “e.g. **222** 🡺 **2**”
  + Find the **offset** of the number. To do that, you can use the formula: (main digit - 2) \* 3
  + If the main digit is **8 or 9**, we need to **add 1** to the **offset**, since the digits **7** and **9** have **4 letters each**
  + Finally, find the **letter index** (a 🡺 0, c 🡺 2, etc.). To do that, we can use the following formula: (offset + digit length - 1).
  + After we’ve found the **letter index**, we can just add that to **the ASCII code** of the lowercase letter “**a**” (97)