Objects and Classes – Exercise

This document defines the lab for the ["C++ OOP" course @ Software University](). Please submit your solutions (source code) to all below-described problems in [Judge]().

Write C++ code for solving the tasks on the following pages.

Code should compile under the C++03 or the C++11 standard.

Please submit a single **`.cpp`** file for each task.

`.cpp` files for the tasks should be named with the task number followed by what you feel describes the exercise in a few words.

E.g. a good name for task 2 of this homework would be:
`2.distance.cpp`

Don't worry about the name too much, just make sure the number and the file extension are correct.

Tasks 4 in this homework require you to be creative about the implementation. It simulates real-world examples of non-strict and sometimes vague descriptions of client requirements. Part of the exercise is to learn to convert paragraphs of text into classes that solve the needs described in the paragraphs and to write code that is easily modified if, for example, the format of the input/output data changes slightly. Also note that some of these exercises mention e.g. "array" or "string" and so on – do not take these literally, you can use any data structures you find appropriate.

I. Homework

1. Sentence Shifter

You are given a **list of words** in one line. On the other line, you are given a **simple integer**.

Your role is to **shift the words** in the sentence **according to that integer**.

For an example, if a sentence has 10 words and you receive a shift number 2 - the first word should become the third, the second word should become the fourth and so on, ..., the word before the last should become the first and the last word should become the second.

Implement this task with a class that is initialized with a **linear container** (array, vector, etc.) of words and which has a **getShiftedSentence()** method which returns the words shifted.

Each word is printed on a different line.

Examples

Input	Output

`Welcome to SoftUni and have fun learning programming`
`2`
	`learning`
	`programming`
	`Welcome`
	`to`
	`SoftUni`
	`and`
	`have`
	`fun`

2. Distance

Write a program to calculate the (Euclidean) distance between two points **p₁** {**x₁**, **y₁**} and **p₂** {**x₂**, **y₂**}. You should write a class to represent such points and a method in it that calculates the distance from the point to another point.

Examples

Input	Output

`3 4`
`6 8`
	`5.000`

`3 4`
`5 4`
	`2.000`

`8 -2`
`-1 5`
	`11.402`

3. Sales

Write a class `Sale` holding the following data: **town**, **product**, **price**, **quantity**. Read a **list of sales** and calculate and print the **total sales by the town** as shown in the output. Order the towns **alphabetically** in the output.

Examples

Input	Output	Comments

`5`
`Sofia beer 1.20 160`
`Varna chocolate 2.35 86`
`Sofia coffee 0.40 853`
`Varna apple 0.86 75.44`
`Plovdiv beer 1.10 88`
	`Plovdiv -> 96.80`
	`Sofia -> 533.20`
	`Varna -> 266.98`
		Plovdiv -> 1.10 * 88 = 96.80
		Sofia -> 1.20 * 160 + 0.40 * 853 = 533.20
		Varna -> 2.35 * 86 + 0.86 * 75.44 = 266.98

4. Total average of students

Write a program, use a class that has params:

- Student Name
- Student Surname
- Total Average

The class should have **print** method that for a given object prints all the information.
Create a vector in `main()` that for a given number (passed thru user) saves the objects
Make a function that calculates the Total **average** of **all** students.
If there are no students, print "`Invalid input`".

Explanation:

Number of students – 2

Name – Maria

Surname – Ivanova

Average – 3.5

Name – Dragan

Surname – Ivanov

Average – 4.5

TOTAL AVERAGE – (3.5 + 4.5) / 2 = 4

Examples

Input	Output
2 Maria Ivanova 3.5 Dragan Ivanov 4.5	Maria Ivanova 3.5 Dragan Ivanov 4.5 4

II. Excluded from Homework

5. Memory* (not included in the homework)

You are given a program in a **`MemoryMain.cpp`**, as well as a **`Company.h`** file, that reads information about **companies** and writes it to the console.

Each company has:

- An **id** (an integer between **`0`** and **`255`**)
- A **name** (a **`string`** containing a sequence of lowercase English letters **`a-z`**)
- **Employees** by their initials (a **`vector`** of **`pair`**s of characters, containing at most **255** employee initials)

The MemoryMain.cpp file reads the information from the console, as a sequence of byte values, stores those bytes in memory (RAM), and then calls a function named **`readCompaniesFromMemory`**, passing it two parameters:

- a **pointer** to the **first byte** in the memory containing the companies
- an integer indicating the **number of companies** stored in the memory

The memory format of each company is the following:

- the first byte contains the **id** of the company (**`0-255`**)
- the **name** of the company starts from the second byte and ends with a null terminator (the value **`0`**, or **`'\0'`**), i.e. the name of the company is placed in memory the same way a null-terminated C-String would be
- the next byte contains the number of employees the company has (**`0-255`**). Let's call it **`numEmployees`**
- the following **`numEmployees * 2`** bytes contain pairs of initials of the employees, i.e. if the **`numEmployees`** byte is at address **`x`**, then the **first employee's first initial** is at address **`x + 1`**, and their **second initial** is at address **`x + 2`**, the **second employee's first initial** is at address **`x + 3`** and their **second** is at address **`x + 4`** and so on.

The **`MemoryMain.cpp`** file will print the companies in the format:

- company **id**, space, company **name**, space, opening bracket **`'('`**, first initial of first employee, dot **'.'**, second initial of first employee, dot **`'.'`**, first initial of second employee, … , closing bracket **`')'`**

For example, if we have the companies:

- **`id = 42, name = "uni", employees = { {'I', 'K'}, {'S', 'N'} }`** and
  `id = 13, name = "joro", employees = { {'G', 'G' } }`

Their representation as **string**s printed by **MemoryMain.cpp** will be:

```
42 uni (I.K.,S.N.)
13 joro (G.G.)
```

Their representation in memory, assuming the memory starts at byte address **M,** will be:

Offset from start	0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13
Value	42	'u'	'n'	'i'	'\0'	2	'I'	'K'	'S'	'N'	13	'j'	'o'	'r'

Offset from start	+14	+15	+16	+17	+18
Value	'o'	'\0'	1	'G'	'G'

And their representation in the input for the task will be:

```
42 117 110 105 0 2 73 75 83 78

13 106 111 114 111 0 1 71 71

end
```

Your task is to create a file called **CompanyMemoryUtils.h** (which **MemoryMain.cpp** includes), containing the function **readCompaniesFromMemory**, implemented in such a way that MemoryMain.cpp compiles successfully and works as described above – i.e. your task is to read the memory, which will be in the format described above, and return a **vector<Company>** containing the companies that were written in that memory.

You should submit a single **.zip** file for this task, containing ONLY the **CompanyMemoryUtils.h** file. The Judge system has a copy of the other files and will compile them along with your **CompanyMemoryUtils.h** file in the same directory.

NOTE: you are also given the code for the test generator used to generate the tests in the judge system, in C#. Compiling and running it will produce random tests (**.in.txt** input files and **.out.txt** output files) similar to those in the Judge system, which you can use to test your code locally.

Examples

Input	Output
42 117 110 105 0 2 73 75 83 78 13 106 111 114 111 0 1 71 71 end	42 uni (I.K.,S.N.) 13 joro (G.G.)
188 105 99 121 104 97 0 3 66 81 72 80 70 83 58 117 97 100 101 108 0 3 83 65 67 72 76 84 end	188 icyha (B.Q.,H.P.,F.S.) 58 uadel (S.A.,C.H.,L.T.)

6. Splender** (not included in the homework)

Splender is a depressed robot who heals his depression by partying and drinking alcohol. To save him from a life of debauchery, his creators have reprogrammed the control system with a more rudimentary intelligence. Unfortunately, he has lost his sense of humor and his former friends have now rejected him.

Splender is now all alone and is wandering through the streets of Futurama with the intention of ending it all in a suicide booth.

To intercept him and save him from almost certain death, the authorities have given you a mission: write a program that will make it possible to foresee the path that Splender follows. To do so, you are given the logic for the new intelligence with which Splender has been programmed as well as a map of the city.

Rules

The 9 rules of the new Splender system:

1) Splender starts from the place indicated by the **@** symbol on the map and heads SOUTH.

2) Splender finishes his journey and dies when he reaches the suicide booth marked $.

3) Obstacles that Splender may encounter are represented by **#** or **X**.

4) When Splender encounters an obstacle, he changes direction using the following priorities: **SOUTH**, **EAST**, **NORTH,** and **WEST**. So he first tries to go **SOUTH**, if he cannot, then he will go EAST, if he still cannot, then he will go **NORTH**, and finally, if he still cannot, then he will go **WEST**.

5) Along the way, Splender may come across path modifiers that will instantaneously make him change direction. The **S** modifier will make him turn **SOUTH** from then on, **E**, to the **EAST**, **N** to the **NORTH,** and **W** to the **WEST**.

6) The circuit inverters (**I** on map) produce a magnetic field which will reverse the direction priorities that Splender should choose when encountering an obstacle. Priorities will become **WEST**, **NORTH**, **EAST**, **SOUTH**. If Splender returns to an inverter I, then priorities are reset to their original state (**SOUTH**, **EAST**, **NORTH**, **WEST**).

7) Splender can also find a few beers along his path (**B** on the map) that will give him strength and put him in "Breaker" mode. Breaker mode allows Splender to destroy and automatically pass through the obstacles represented by the character **X** (only the obstacles **X**). When an obstacle is destroyed, it remains so permanently and Splender maintains his course of direction. If Splender is in Breaker mode and passes over a beer again, then he immediately goes out of Breaker mode. The beers remain in place after Splender has passed.

8) 2 teleporters **T** may be present in the city. If Splender passes over a teleporter, then he is automatically teleported to the position of the other teleporter and he retains his direction and Breaker mode properties.

9) Finally, the space characters are blank areas on the map (no special behavior other than those specified above).

Your program must display the sequence of moves taken by Splender according to the map provided as input.

The map is divided into lines (**L**) and columns (**C**). The contours of the map are always unbreakable **#** obstacles. The map always has a starting point **@** and a suicide booth **$**.

Let the map below:

```
######
#@E $#
# N  #
#X   #
```

```
#######
```

In this example, `Splender` will follow this sequence of moves:

SOUTH (initial direction)

EAST (because of the obstacle **X**)

NORTH (change of direction caused by **N**)

EAST (change of direction caused by **E**)

EAST (current direction, until end point **$**)

Examples

Input	Output
`10 10`	`SOUTH`
`##########`	`SOUTH`
`#        #`	`EAST`
`#  S   W #`	`EAST`
`#        #`	`EAST`
`#  $     #`	`EAST`
`#        #`	`EAST`
`#@       #`	`EAST`
`#        #`	`NORTH`
`#E     N #`	`NORTH`
`##########`	`NORTH`
	`NORTH`
	`NORTH`
	`NORTH`
	`WEST`
	`WEST`
	`WEST`
	`WEST`
	`SOUTH`
	`SOUTH`
`5 5`	`SOUTH`
`#####`	`SOUTH`
`#@  #`	`EAST`
`#   #`	`EAST`
`#  $#`	
`#####`	
`8 8`	`SOUTH`
`########`	`EAST`
`# @    #`	`EAST`
`#     X#`	`EAST`
`# XXX  #`	`SOUTH`
`#   XX #`	`EAST`
`#   XX #`	`SOUTH`
`#     $#`	`SOUTH`
`########`	`SOUTH`