# The Fight for Gondor

**You can test your solutions in** [**Judge**](https://judge.softuni.org/Contests/Practice/Index/2813#0)**.**

*Sauron’s army is marching towards Gondor. Vicious waves of orcs are getting ready to attack Aragorn’s people and make their way into the city.*

**First**, you will be given a **number** equal to the **waves of orcs**. On the **second** line, you will be given the **plates of the Aragorn's defense**. Then, on each next line (**for each wave**), you receive the power of **each orc warrior**. Additionally, on every **third wave**, the people build a **new plate** (**extra** line with a single integer) **before** the Sauron**'**s warriors attack. To enter the city, the orcs have to **destroy all the plates**.

**Until** there are **no more plates** or **orcs**, the **last orc warrior** attacks **the first plate**:

* If the **warrior's** value is **greater**, he **destroys** the plate and **lowers** his value by the plate**'**s value, then attacks the **next** plate, **until** his value reaches 0.
* If the **plate's** value is **greater**, the warrior **dies** and the plate **decreases** its value by the warrior**'**s value.
* If their values are **equal**, the warrior **dies** and the plate is **destroyed**.

### Input

* **First** line: integer- the number of **waves**
* **Second** line: integers, representing the **plates**, **separated by a single space**.
* For each **wave:** integers, representing the **warrior orcs**, **separated by a single space**.
  + On every **third** wave, you will be given an **extra line** with a **single** integer, which will be the **plate you need to add**. **[!]** Add the plate **before** processing the attacks. **[!]**

### Output

* On the first line of output – print if the orcs destroyed the Gondor**'**s defense:
  + True: "**The orcs successfully destroyed the Gondor's defense.**"
  + False: "**The people successfully repulsed the orc's attack.**"
* On the second line - print all plates or orcs left, separated by comma and a white space:
  + If there are warriors: "**Orcs left: {orc1}, {orc2}, {orc3},** **…**"
  + If there are plates: "**Plates left: {plate1}, {plate2}, {plate3}, …**"

### Constraints

* All of the given numbers will be valid integers in the range [1, 100].
* **Not all waves** may be needed to destroy the defense.
* There will **always** be a **winning side**, meaning either no orcs or plates left.

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **3**  **10 20 30**  **4 5 1**  **10 5 5**  **10 10 10**  **4** | **The people successfully repulsed the orc's attack.**  **Plates left: 4** | * The first wave (4 5 1):   + Orc (1) attacks Plate (10) => dies and plate is now 9.   + Orc (5) attacks Plate (9) => dies and the plate is now 4.   + Orc (4) attacks Plater (4) => dies and the plate is gone. * The second wave (10 5 5):   + Orc (5) attacks Plate (20) => dies and plate is now 15.   + Orc (5) attacks Plate (15) => dies and plate is now 10.   + Orc (10) attacks Plate (10) => dies and plate is gone. * The third wave (10 10 10):   + People build a new plate (4), plates are now: 30 4   + Orc (10) attacks Plate (30) => dies and plate is now 20.   + Orc (10) attacks Plate (20) => dies and plate is now 10.   + Orc (10) attacks Plate (10) => dies and plate is gone. * We have no more waves and one plate left (4) => see the output. |
| **5**  **10 30 10**  **3 3 4**  **10 10 10**  **5 5**  **5**  **7 6**  **8 6 7** | **The orcs successfully destroyed the Gondor's defense.**  **Orcs left: 1, 7** | * The first wave (3 3 4):   + Orc (4) attacks Plate (10) => dies and the plate is now 6.   + Orc (3) attacks Plate (6) => dies and the plate is now 3.   + Orc (3) attacks Plater (3) => dies and the plate is gone. * Second wave (10 10 10):   + Orc (10) attacks Plate (30) => dies and plate is now 20.   + Orc (10) attacks Plate (20) => dies and plate is now 10.   + Orc (10) attacks Plate (10) => dies and plate is gone. * Third-wave (5 5):   + People build a new plate (5), plates are now: 10 5   + Orc (5) attacks Plate (10) => dies and the plate is now 5.   + Orc (5) attacks Plate (5) => dies and the plate is gone. * Fourth wave (7 6):   + Orc (6) attacks Plate (5) => the orc is now 1 and the plate is gone. * We have no more plates, so the waves stop coming => see the output. Also, we stop the input. (8 6 7 has not proceeded, but is in the input because the waves are 5) |

# 02. Super Mario

**You can test your solutions in** [**Judge**](https://judge.softuni.org/Contests/Practice/Index/2824#1)**.**

*Princess Peach is imprisoned in the biggest tower. It’s up to Mario to save her.*

After Mario gets into the tower, he has to fight his way to the princess. To do that, he has to walk through the maze where the dangerous Bowser is guarding, but he also has to be careful not to lose all his lives and not be able to proceed with his mission. If Mario successfully reaches the throne, he saves Princess Peach.

The castles maze may look like this:

|  |  |
| --- | --- |
| **The Maze** | **Legend** |
| ------P--- -------B-- --B------- ---------- ----M----- | **M** 🡺 **Mario**, **the player character**  B🡺 **Bowser, the enemy**  P 🡺 **Princess Peach**  - 🡺 **Empty space** |

Each turn proceeds as follows:

* **First**, Bowser **spawns** on the given index.
* **Then, Mario** moves in a direction, which **decreases** his life by 1.
  + It can be "**W**" (up), "**S**" (down), "**A**" (left), "**D**" (right).
  + If Mario tries to move **outside** of the maze, he **doesn’t** move but **still** has his life **decreased**.
* If an enemy is in the **same cell** where Mario moves, Bowser fights him, which **decreases** his life by 2. If Mario’s lives **drop** at 0 or below, he **dies** and you should mark his position with ‘**X**’.
* If Mario **kills** the enemy successfully, the enemy **disappears**.
* If Mario reaches the **index** where **the throne** is, **he saves Princess Peach and they both run away** (disappear from the field)**, even if his lives are 0 or below.**

## Input

* On the **first line** of input, you will receive **e** – **the lives** Mario has.
* On the **second line** of input, you will receive **n** – the **number of rows** the castle’s maze will consist of.  
  Range: **[5-20]**
* On the next **n lines**, you will receive how each row looks.
* Then, **until** Mario dies, or reaches the princess, you will receive a **move command** and **spawn row and column**.

## Output

* If Mario **runs out of li****fe**, print "Mario died at {row};{col}."
* If Mario **reaches the throne**, print "Mario has successfully saved the princess! Lives left: {lives}"
* Then, in all cases, **print** the **final state of the field** on the **console**.

## Constraints

* The field will always be **rectangular**.
* Mario will **always** run out of life or reach the throne.
* There will be **no case** with spawn on an **invalid** index.
* There will be **no case** with **two enemies on the same cell**.
* There will be **no case** with enemy **spawning** on the index **where Mario or the princess is**.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 100  5  --P--  -----  -----  -----  --M--  W 3 0  W 3 1  W 3 2  W 3 3 | Mario has successfully saved the princess! Lives left: 96  -----  -----  -----  BBBB-  ----- | Turn 1: An enemy spawns at [3;0], Mario moves to [3;2], his lives decrease by 1.  Turn 2: An enemy spawns at [3;1], Mario moves to [2;2], his lives decrease by 1.  Turn 3: An enemy spawns at [3;2], Mario moves to [1;2], his lives decrease by 1.  Turn 4: An enemy spawns at [3;3], Mario moves to [0;2], his lives decrease by 1, but he also moves to the index where the princess is – they run away. |
| 3  5  --P--  -----  -----  -----  --M--  W 3 2 | Mario died at 3;2.  --P--  -----  -----  --X--  ----- | Turn 1: An enemy spawns at [3;2], Mario moves to [3;2], his lives decrease by 1, and fights the enemy at that index. Mario’s life is decreased by 2, dropping it to 0 or below => Mario dies. |

# 03. Classroom

**You can test your solutions in** [**Judge.**](https://judge.softuni.org/Contests/Practice/Index/2597#2)

## Preparation

Download the skeleton provided in Judge. **Do not** change the **StartUp** class or its **namespace**.

## Problem description

Your task is to create a repository, which stores items by creating the classes described below.

First, write a **C#** class **Student** with the following properties:

* **FirstName: string**
* **LastName: string**
* **Subject: string**

The class **constructor** should receive **firstName, lastName** and **subject.** You need to create the appropriate **getters and setters**. The class should override the **ToString()** method in the following format:

**"Student: First Name = {firstName}, Last Name = {lastName}, Subject = {subject}"**

**Next**, write a **C#** class **Classroom** that has **students** (a collection, which stores the **students**) and a certain capacity. All entities inside the repository have the **same fields**. Also, the **Classroom** class should have the following properties:

* **Capacity: int**
* **Count: int – returns the number of students in the classroom**

The class **constructor** should receive **capacity**, also it should initialize the **students** with a new instance of the collection. Implement the following features:

* Field **students** – a **collection** that holds added students
* Method RegisterStudent(Student student) – **adds** an **entity** to the students **if** **there** **is** an **empty seat** for the student.
  + Returns **"Added student {firstName} {lastName}"** if the student is successfully added
  + Returns **"No seats in the classroom" –** if there are no more seats in the classroom
* Method DismissStudent(string firstName, string lastName) – removes the student by **the given names**
  + Returns **"Dismissed student {firstName} {lastName}"** if the student is successfully dismissed
  + Returns **"Student not found"** if the student is not in the classroom
* Method **GetSubjectInfo(string subject)** – returns all the students with the **given subject in the following format:**

**"Subject: {subjectName}  
Students:  
{firstName} {lastName}  
{firstName} {lastName}  
…"**

* + Returns **"No students enrolled for the subject"** if the student is not in the classroom
* Method GetStudentsCount() – returns the **count** of the **students in the classroom**.
* Method **GetStudent(string firstName, string lastName)** – returns the student with the given names.

## Constraints

* The **combinations** of **names** will **always be unique**.
* The **capacity** will always be **a positive** **number**.

## Examples

This is an example of how the **Classroom** class is **intended to be used**.

|  |
| --- |
| Sample code usage |
| // Initialize the repository  Classroom classroom = new Classroom(10);  // Initialize entities  Student student = new Student("Peter", "Parker", "Geometry");  Student studentTwo = new Student("Sarah", "Smith", "Algebra");  Student studentThree = new Student("Sam", "Winchester", "Algebra");  Student studentFour = new Student("Dean", "Winchester", "Music");  // Print Student  Console.WriteLine(student); // Student: First Name = Peter, Last Name = Parker, Subject = Geometry  // Register Student  string register = classroom.RegisterStudent(student);  Console.WriteLine(register); // Added student Peter Parker  string registerTwo = classroom.RegisterStudent(studentTwo);  string registerThree = classroom.RegisterStudent(studentThree);  string registerFour = classroom.RegisterStudent(studentFour);  // Dismiss Student  string dismissed = classroom.DismissStudent("Peter", "Parker");  Console.WriteLine(dismissed); // Dismissed student Peter Parker  string dismissedTwo = classroom.DismissStudent("Ellie", "Goulding");  Console.WriteLine(dismissedTwo); // Student not found  // Subject info  string subjectInfo = classroom.GetSubjectInfo("Algebra");  Console.WriteLine(subjectInfo);  // Subject: Algebra  // Students:  // Sarah Smith  // Sam Winchester  string anotherInfo = classroom.GetSubjectInfo("Art");  Console.WriteLine(anotherInfo); // No students enrolled for the subject  // Get Student  Console.WriteLine(classroom.GetStudent("Dean", "Winchester"));  // Student: First Name = Dean, Last Name = Winchester, Subject = Music |

## Submission

Zip all the files in the project folder except **bin** and **obj** folders

Submit a **single .zip file**, containing the **Classroom package, with the classes inside (Student, Classroom, and the StartUp class)**. There is no specific content required inside the **StartUp** class e. g. you can do any kind of local testing of your program there. However, there should be **Main(string[] args)** method inside.