# Problem 2 – Parking Feud

Sam finally got through the **crossroads** safely and made it to the airport. Now, he needs to **park** his **car** at the premium airport **parking** **lot**, so it’s safe and sound when he returns from his vacation. He has already pre-paid for his parking spot, so we needn’t worry about that. He even got a 20% discount for the early payment.

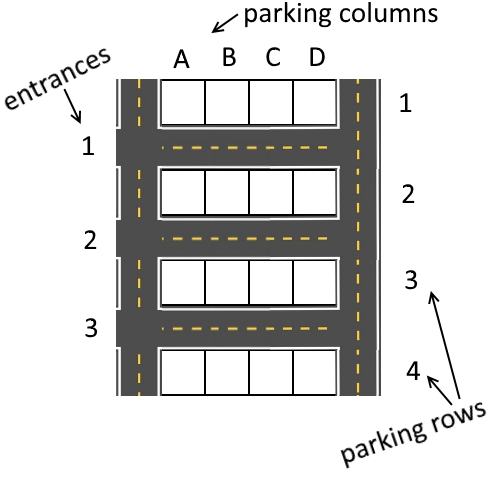
Anyway, the **parking** **lot**, as you might’ve deduced, is a **rectangular** **matrix**. The **first** and the **last** column of it will **always** **be** **empty** and the **first** **column** will also serve as an **entrance** to the **parking** **lot**. Starting from the **first** **row** (**included**), **every** **other** **row** will consist of parking spots. Parking spot rows will **always** be an **odd** **number**, so the **top** and **bottom** **rows** will **always** be parking rows.

Although Sam was stuck in traffic, the parking lot has just opened, so all the spots will be free when he arrives. He must go to an entrance, receive a parking spot location (represented by a **capital** **Latin** **letter** and a **number**), navigate to it and park his car. **Simple** enough. The problem is that the parking company is having an internet issue and thus, **no** **communication** is possible between the **different** **entrances**. To make Sam’s day even better, they might give **somebody** **else** the **very** **same** **parking** **spot**.

Your job is to help Sam **solve** these **conflicts** and **calculate** **how** **much** **distance** (number of squares) he has traveled before parking successfully.

* You will receive the **number** of **parking** **columns** and **rows** and will have to **calculate** the **size** of and **create** a **parking** **lot** (matrix).
* Then, upon approaching the parking, Sam will choose an **entrance** (numbered from 1 to **n**, where **n** is **always** 1 **less** than the **number** of **parking** **rows**). Sam will use this **entrance** **row** to **enter** the **parking** lot and will **always** **return** to it after an **unsuccessful** **attempt** to **park**.
* To park at a spot, Sam needs to **move** to a **road** **square** right next to the **desired** **spot**.
  + If Sam needs to get to a **different** **row**, he needs to **pass** **through** the **whole** **row** he’s on, go **2 squares up or down** the **free** **column** and move on the **next** **row**.
  + Sam can **NOT** **skip** **rows**, even if he needs to **move** 2 or **more** **rows** **away** from his **current** **one**. He can **NOT** drive through parking spots, even if they are empty.
  + Sam starts on the **left** on his **entrance** **row** and **moves** to the **right**. After **navigating** to a **row** **below**/**above** he will be on the **right** and **move** **left** and so on with **every** **following** row, in a **zig**-**zag** **manner**.
  + **Other** **drivers** move the **same** **way** as Sam does.
* If there is no conflict, Sam will **drive** to his **targeted** parking **spot** and **park** **successfully**.
* If a conflict occurs, you need to **calculate** which **driver** will **pass** **less** **squares** to get to that **spot**.
  + Sam will **always** **drive** to that **spot**, but if the **other** **driver** is **closer**, it will be **taken** when Sam reaches it. If the spot is **taken**, Sam **returns** to **his** **entrance**, **doubling** his **passed** **distance** on the **current** **turn**.
  + If Sam and another driver are at the **same** **distance**, Sam will **speed** **up** and **take** the **spot**, winning the conflict and **parking** **successfully**.
* Your program **ends** when Sam **parks** **successfully**.

## Input

* On the **first line** of input, you will receive **2 numbers**: the **parking spot rows** and **parking spot columns**. Those are **NOT** the **actual** **matrix** **dimensions**. For example a 4 parking spot rows and 4 parking spot columns matrix would look like this:  
  
* On the next **line**, you will receive the **number** of the **entrance** at which Sam will enter the parking lot. Entrances will be numbered as shown on the picture above.
* On the **following lines**, until Sam **parks** **successfully**, you will receive a row of **parking** **spots**, one for each entrance.

## Output

* When Sam **parks** **successfully**, **end** the **program** and **print** on the console on **two** **rows**:
  + The **spot** Sam has parked at in the format:  
    "Parked successfully at {parkingSpot}."
  + The distance Sam passed in the format:  
    "Total Distance Passed: {totalDistance}"

## Constraints

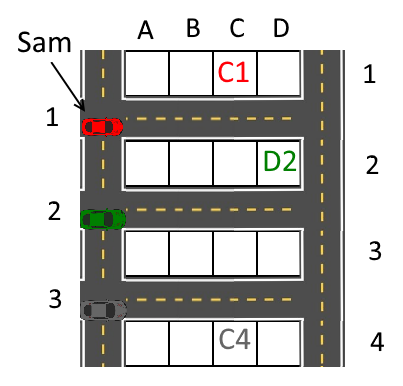
* The parking will always be **rectangular**.
* Conflicts will **only** occur between **Sam** and **another** **driver**
* There will **never** be **3** cars in a conflict
* There will **never** be an **invalid** **parking** **spot** given as **input**
* There will **never** be a taken **parking** **spot** given as **input**
* The **parking** **spots** in the **input** will **always** be the same number as the entrances
* The parking lot will always have at least 3 **parking** **rows** and 1 **column**
* The parking lot will **not** have more than **10** **parking** **rows** and **columns**

## Examples

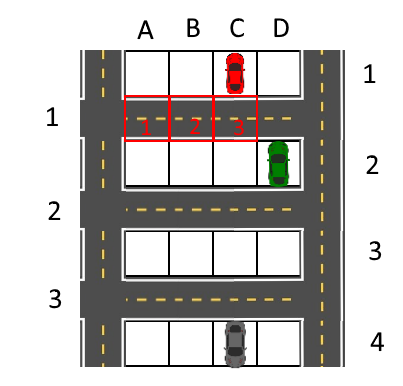
First, let’s look at a simple parking without any conflicts.

|  |
| --- |
| **Input** |
| 4 4  1  C1 D2 C4  A1 D2 C1 |
| 4 4  1  D2 D1 C4  A1 D2 C1 |

Sam **enters** the parking lot at **entrance** **number** **1** and has to navigate to spot **C1**:



Then, he **moves** through the **3 squares marked in red** and **parks** **successfully** at **C1**:



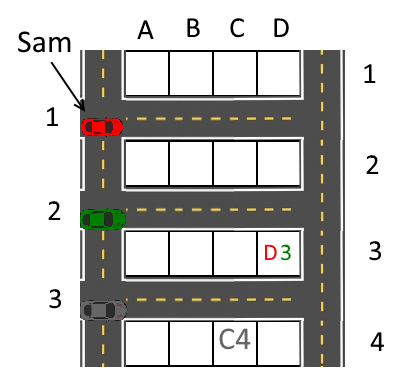
|  |
| --- |
| **Output** |
| Parked successfully at C1.  Total Distance Passed: 3 |

Keep in mind, that you should end your program and not read the last row of input!

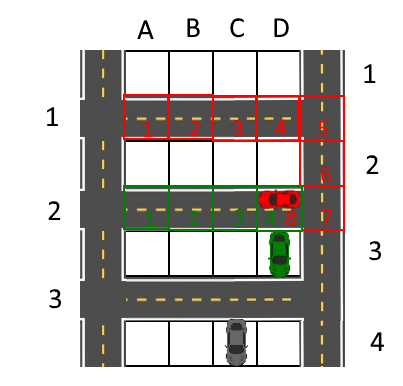
Next, let’s look at an example with a conflict:

|  |
| --- |
| **Input** |
| 4 4  1  D3 D3 C4  A1 D2 C1  4 4  1  D3 D2 C4  A1 D2 C1 |

We can see that **both** Sam and the **driver** at **entrance** **2** have the **same** parking **spot**:



The green car is only 4 squares away from the spot, while Sam is 8 squares away, so the green car takes the spot and Sam will have to return to the start, passing a total of 16 squares.



Then, Sam receives **another** **spot** with no conflict and parks successfully.

|  |
| --- |
| **Output** |
| Parked successfully at A1.  Total Distance Passed: 17 |

And finally, a more **complicated** example, to help you **test** your **program**:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 4  4  C1 B2 D3 B2  B1 A1 D2 D2  D1 C3 D5 D5 | Parked successfully at D5.  Total Distance Passed: 72 |