Lab 5 D

Problem 1: The Great Cake Dilemma

During the Bake Festival, two chefs, Chef Whisk and Chef Spoon, always bake cakes and share it to the people in town. They believe all the people who are given cakes should get equal number of slices. But they have a quirky problem: they always bake cakes with a different number of slices. They want to find the largest number of people they can share their cakes with such that the cakes can be evenly shared leaving any leftovers. Your task is to help the chefs figure out the largest number of people that can eat cakes everyday!

Input Format:

- The first line contains an integer `N`, representing the number of days.
- The next `N` lines each contain two integers: `s1` (slices in Chef Whisk's cake) and `s2` (slices in Chef Spoon's cake).

Constraints:

- $1 \leq$ `N` ≤ 100
- $1 \leq$ `s1`, `s2` $\leq 10^{18}$

Output Format:

Return a single integer: the largest number of people that was served during the festival

Input 1:

```
 3
12 15
18 24
24 36
```

Copy

Output 2:

```
12
```

Copy

Input 2:

```
2
7 5
13 27
```

Copy

Output 2:

```
1
```

Copy

Explanation:

In the first example, the maximum number of slices that can be evenly divided between Chef Whisk and Chef Spoon for each pair of cakes are:

- For the pair (12, 15), they can serve 3 people, with each person getting 4 slices from the first cake and 5 from the second cake.
- For the pair (18, 24), they can serve 6 people, with each person getting 3 slices from the first cake and 4 from the second cake.
- For the pair (24, 36), they can serve 12 people, with each person getting 2 slices from the first cake and 3 from the second cake.

The largest number of people they can serve during the festival is `12`.

In the second example, they can only serve 1 from both pairs, so the output is `1`.

Problem 2 : Generate All Valid Parentheses Combinations

You are given an integer **n**, representing the number of pairs of parentheses. Your task is to generate all possible valid combinations of **n** pairs of parentheses, where a valid combination ensures every opening parenthesis `(` has a corresponding closing parenthesis `)` and that they are properly nested.

Use a recursive approach to solve the problem, and generate the combinations by ensuring the following:

- You cannot add more closing parentheses than the number of opening parentheses at any point.
- When the length of the current combination reaches `2 * n`, a valid combination has

been formed.

Input Format:

- The input consists of a single integer **n**, where **n** represents the number of pairs of parentheses.

Output Format:

Print total number of valid combinations of parentheses. Print each valid combination on a new line.

Constraints:

- 1 ≤ n ≤ 10
- The total number of valid combinations should be printed in any order.

Input 1:

```
3
```
Copy

Output:

```
((()))
(()())
(())()
()(())
()()()
```
Copy

Explanation:

These are valid combinations of parentheses, output order does not matter.

Problem 3: Vector Multiplication - A Galactic Conundrum

In a distant galaxy far, far away, the interstellar travelers from planet **Mathlore** are facing a peculiar problem. They are navigating through space and have encountered two asteroids, each represented by vectors in 3D space. These space travelers, known for their love of mathematical challenges, must figure out if these asteroids will collide or pass by harmlessly. To solve this, they need your help in calculating the **cross product** of the asteroids' vectors and determine whether the paths are **parallel** to each other.

The interstellar navigators have just enough energy to run your code once, so they need you to define a struct in C that represents the x, y, and z components of the vectors. Help them compute the cross product of these vectors, and figure out if the asteroids are heading for an intergalactic showdown (parallel paths) or just a friendly flyby.

Task 1: Compute the Cross Product

When given two vectors, say `A` and `B`, the cross product will give the travelers a new vector that is perpendicular to both. This is essential to avoid *space dust* in their navigational systems.

Display the cross product so the travelers can recalibrate their trajectory.

Task 2: Check for Parallel

Once the cross product is calculated, the travelers must determine if the vectors are parallel. If parallel, it means the asteroids are on a direct collision course! Otherwise, it’s smooth sailing ahead.

Input Format:

- The input consists of two vectors, each with three integer components (x, y, z).

Output Format:

- Display the cross product of the two vectors. (format - 3 integers)
- Indicate if the vectors are parallel, or if the travelers are safe. (format - YES/NO)

Constraints:

- The vector components must be integers between -1000 and 1000.

Example Input 1:

```
3 -3 1
4 9 2
```

Example Output 1:

```
-15i -2j +39k
NO
```

Example Input 2:

```
1 0 0
0 1 0
```

Example Output 2:

```
k
NO
```

Example Input 3:

```
0 0 0
0 0 0
```

Example Output 3:

```
0
YES
```

Submission Guidelines

Do not rename any files given in the handout. Only write the code in the specified C files in the respective directories.