Co-prime pairs in C++

```
#include <iostream>
using namespace std;

class CoPrimePairs {
public:
    static void main() {
        int n = 10;

        for (int i = 0; i < n / 2; i++) {
            cout << 2 * i + 1 << " " << 2 * i + 2 << endl;
        }
    }
};

int main() {
    CoPrimePairs::main();
    return 0;
}
```

Dry Run Table for n = 10

i	2*i + 1	2*i + 2	Output
0	1	2	1 2
1	3	4	3 4
2	5	6	5 6
3	7	8	7 8
4	9	10	9 10

🖶 **Output**

```
1 2
3 4
5 6
7 8
9 10
```

```
1 2

3 4

5 6

7 8

9 10
```

GCD array in C++

```
#include <iostream>
#include <vector>
using namespace std;

// Function to compute GCD of two numbers using Euclidean algorithm
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

// Function to compute GCD of an array of integers
int gcdArray(vector<int>& arr) {
    int result = arr[0];
    for (int i = 1; i < arr.size(); i++) {
        result = gcd(result, arr[i]);
        if (result == 1) { // If result becomes 1, further GCD will also be 1
            return 1;
        }
    }
    return result;
}

int main() {
    vector<int> arr = {12, 24, 36, 48};
    cout << "GCD of the array elements: " << gcdArray(arr) << endl;
    return 0;
}
```

Step-by-Step Dry Run (Tabular Form)

We'll use this table to track the intermediate GCD results:

Step	`result` (previous GCD)	`arr[i]`	`gcd(result, arr[i])`
1	12	24	gcd(12, 24) = 12
2	12	36	gcd(12, 36) = 12
3	12	48	gcd(12, 48) = 12

Since the GCD never drops to 1, we never hit the `if (result == 1)` shortcut.

📌 Final Output:

```
GCD of the array elements: 12
```

GCD of the array elements: 12

NumberofSubArrayswithGCDequaltoK **in C++**

```
#include <iostream>
#include <vector>
using namespace std;

class NumberofSubArrayswithGCDequaltoK {
public:
    int subarrayGCD(vector<int>& nums, int k) {
        int count = 0;
        int n = nums.size();

        for (int sp = 0; sp < n; sp++) {
            int ans = 0;
            for (int ep = sp; ep < n; ep++) {
                ans = gcd(ans, nums[ep]);

                if (ans < k) {
                    break;
                }
                if (ans == k) {
                    count++;
                }
            }
        }

        return count;
    }

    int gcd(int a, int b) {
        if (a == 0) {
            return b;
        }
        return gcd(b % a, a);
    }
};

int main() {
    NumberofSubArrayswithGCDequaltoK solution;

    // Hard-coded input
    vector<int> nums = {2, 4, 6, 8, 3, 9};
    int k = 3;

    int result = solution.subarrayGCD(nums, k);
    cout << "Number of subarrays with GCD equal to "
<< k << ": " << result << endl;

    return 0;
}
```

Input:

nums = {2, 4, 6, 8, 3, 9}
k = 3

We'll check **all subarrays** and see how many have GCD = 3.

📊 Dry Run Table

sp	Subarray	ans (GCD)	Matches k?
0	[2]	2	✗
0	[2, 4]	2	✗
0	[2, 4, 6]	2	✗
0	[2, 4, 6, 8]	2	✗
0	[2, 4, 6, 8, 3]	1	✗ (GCD < k) – break
1	[4]	4	✗
1	[4, 6]	2	✗
1	[4, 6, 8]	2	✗
1	[4, 6, 8, 3]	1	✗ (GCD < k) – break
2	[6]	6	✗
2	[6, 8]	2	✗
2	[6, 8, 3]	1	✗ (GCD < k) – break
3	[8]	8	✗
3	[8, 3]	1	✗ (GCD < k) – break
4	[3]	3	✓
4	[3, 9]	3	✓
5	[9]	9	✗

✅ Final Count

We found **2 subarrays** where the GCD is exactly 3:

- [3]
- [3, 9]

🧠 Explanation of Logic

You're using a **nested loop**:

- Outer loop: start point sp
- Inner loop: end point ep
- You maintain a running GCD of the subarray
- If GCD < k, you **break** early (smart optimization)
- If GCD == k, increment the counter

	And your GCD function is correct, based on the Euclidean algorithm. 🧪 **Output:** Number of subarrays with GCD equal to 3: 2
Number of subarrays with GCD equal to 3: 2	

Subsequence with GCD in C++

```
#include <iostream>
using namespace std;

class SubsequencewithGCD {
public:
    static void main() {
        int arr[] = {1, 2, 3, 4};
        int n = sizeof(arr) / sizeof(arr[0]);

        int ans = 0;
        for (int i = 0; i < n; i++) {
            ans = gcd(ans, arr[i]);
        }

        if (ans == 1) {
            cout << "true" << endl;
        } else {
            cout << "false" << endl;
        }
    }

    static int gcd(int a, int b) {
        if (b == 0) {
            return a;
        } else {
            return gcd(b, a % b);
        }
    }
};

int main() {
    SubsequencewithGCD::main();
    return 0;
}
```

Dry Run on Given Input

arr[] = {1, 2, 3, 4}

Let's compute:

Step	i	arr[i]	Current GCD (ans)
1	0	1	gcd(0, 1) = 1
2	1	2	gcd(1, 2) = 1
3	2	3	gcd(1, 3) = 1
4	3	4	gcd(1, 4) = 1

✅ Final GCD = 1 → So the output will be:

true

✅ Output

true

true