Friend's pairing in C++

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

int counter = 1;

void solution(int i, int n, vector<bool>& used, string
asf) {
    if (i > n) {
        cout << counter << "." << asf << endl;
        counter++;
        return;
    }

    if (used[i]) {
        solution(i + 1, n, used, asf);
    } else {
        used[i] = true;
        solution(i + 1, n, used, asf + "(" + to_string(i) + ")
");
        for (int j = i + 1; j <= n; j++) {
            if (!used[j]) {
                used[j] = true;
                solution(i + 1, n, used, asf + "(" + to_string(i)
+ "," + to_string(j) + ") ");
                used[j] = false;
            }
        }
        used[i] = false;
    }
}

int main() {
    int n = 3;
    vector<bool> used(n + 1, false);
    solution(1, n, used, "");
    return 0;
}
```

Function Logic Recap

Dry Run for `n = 3`

Step	i	used	Action	Output (if any)
1	1	`[F, F, F, F]`	`1` unused → go alone: `(1)`	
2	2	`[F, T, F, F]`	`2` unused → go alone: `(2)`	
3	3	`[F, T, T, F]`	`3` unused → go alone: `(3)`	`1.(1) (2) (3)`
4			backtrack to pair `2` and `3`	`2.(1) (2,3)`
5			backtrack to try `1` with `2`	
6	2	`[F, T, T, F]`	`3` unused → alone: `(3)`	`3.(1,2) (3)`
7			backtrack	
8			try `1` with `3`	
9	2	`[F, T, F, T]`	`2` unused → alone: `(2)`	`4.(1,3) (2)`

✓ Final Output

```
1.(1) (2) (3)
2.(1) (2,3)
3.(1,2) (3)
4.(1,3) (2)
```

Output:-

1.(1) (2) (3)
2.(1) (2,3)
3.(1,2) (3)
4.(1,3) (2)