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;
}
```

Dry Run

Input: n = 3
Function Call: solution(1, 3, used = {false, false, false, false}, "")

1. i=1 (not used) → Mark 1 used → Call solution(2, 3, used={false, true, false, false}, "(1) ")
    - i=2 (not used) → Mark 2 used → Call solution(3, 3, used={false, true, true, false}, "(1) (2) ")
        - i=3 (not used) → Mark 3 used → Call solution(4, 3, used={false, true, true, true}, "(1) (2) (3) ")
            - Base case: Print 1.(1) (2) (3)
        - Backtrack: Unmark 3.
    - Backtrack: Unmark 2.
    - i=2 → Pair 2,3 → Mark 2,3 used → Call solution(4, 3, used={false, true, true, true}, "(1) (2,3) ")
        - Base case: Print 2.(1) (2,3)
    - Backtrack: Unmark 2,3.

- Backtrack: Unmark 1.

2. i=1 → Pair 1,2 → Mark 1,2 used → Call solution(3, 3, used={false, true, true, false}, "(1,2) ")
    - i=3 (not used) → Mark 3 used → Call solution(4, 3, used={false, true, true, true}, "(1,2) (3) ")
        - Base case: Print 3.(1,2) (3)
    - Backtrack: Unmark 3.

- Backtrack: Unmark 1,2.

3. i=1 → Pair 1,3 → Mark 1,3 used → Call solution(3, 3, used={false, true, false, true}, "(1,3) ")
    - i=2 (not used) → Mark 2 used → Call solution(4, 3, used={false, true, true, true}, "(1,3) (2) ")
        - Base case: Print 4.(1,3) (2)
    - Backtrack: Unmark 2.

- Backtrack: Unmark 1,3.

Output:-

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