Remove Invalid Parenthesis in C++

```
#include <iostream>
#include <string>
#include <unordered_set>
#include <stack>
using namespace std;

void solution(string str, int mra,
unordered_set<string>& ans);
int getMin(string str);

void solution(string str, int mra,
unordered_set<string>& ans) {
   if (mra == 0) {
      int mrnow = getMin(str);
      if (mrnow == 0) {
         if (ans.find(str) == ans.end()) {
            cout << str << endl;
            ans.insert(str);
         }
      }
      return;
   }
   for (int i = 0; i < str.length(); i++) {
      string left = str.substr(0, i);
      string right = str.substr(i + 1);
      solution(left + right, mra - 1, ans);
   }
}

int getMin(string str) {
   stack<char> st;
   for (int i = 0; i < str.length(); i++) {
      char ch = str[i];
      if (ch == '(') {
         st.push(ch);
      } else if (ch == ')') {
         if (st.empty()) {
            st.push(ch);
         } else if (st.top() == ')') {
            st.push(ch);
         } else if (st.top() == '(') {
            st.pop();
         }
      }
   }
   return st.size();
}

int main() {
   string str = "((((()))";
   unordered_set<string> ans;
   int mra = getMin(str);
   solution(str, mra, ans);
   return 0;
}
```

Step-by-Step Dry Run:

Step 1: Calculate mra using getMin(str)

The string is "((((()))", and we need to calculate how many parentheses need to be removed to balance the string.

- **Initial String**: "((((()))"
- Using a stack, we process the string:
  - Encountering an opening parenthesis (: Push onto the stack.
  - Encountering a closing parenthesis): Pop an opening parenthesis from the stack (if one exists).
  - After processing the entire string, we find that 2 opening parentheses (do not have corresponding closing parentheses).
- **Result of getMin("((((()))")**: The minimum number of removals (mra) is **2** because we need to remove 2 redundant opening parentheses (.

Step 2: Recursive Function solution(str, mra, ans)

We now start generating possible valid strings by removing parentheses one by one, up to a total of mra = 2 removals.

- **First Call: solution("((((()))", 2, ans)**:
  - The string has 2 removable parentheses, so we explore all possible ways of removing parentheses.

Recursive Steps:

1. **Remove Parenthesis at index 0 (First (**:
   - String becomes: "(((())"
   - Call solution("(((())", 1, ans).
2. **Remove Parenthesis at index 0 again (First (in "(((())")**:
   - String becomes: "((())"
   - Call solution("((())", 0, ans).
3. **Base Case: solution("((())", 0, ans)**:
   - We check if the string "((())" is balanced using getMin("((())").
   - The result is 0, meaning the string is balanced.
   - Since it is valid and not already in ans, we print it and add it to ans.

   Valid String Output: ((()))

4. **Backtrack to Step 2 and explore other removals**:

	○ We explore other combinations, but in this particular case, only the string "((()))" is valid after removing 2 parentheses. ○ All other combinations generated during recursion either involve invalid strings or are duplicates. **Final Output:** After backtracking through all combinations, the only valid string left is: ((()))
Output:- ((()))	