**Permutation in C++**

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

void permutations(int cb, int nboxes, int items[], int
ssf, int ritems, string asf) {
   if (cb > nboxes) {
      if (ssf == ritems) {
         cout << asf << endl;
      }
      return;
   }

   for (int i = 0; i < ritems; i++) {
      if (items[i] == 0) {
         items[i] = 1;
         permutations(cb + 1, nboxes, items, ssf + 1,
ritems, asf + to_string(i + 1));
         items[i] = 0;
      }
   }

   permutations(cb + 1, nboxes, items, ssf, ritems, asf
+ "0");
}

int main() {
   int nboxes = 3;
   int ritems = 2;
   int cb = 1;
   int ssf = 0;
   int items[ritems] = {0}; // Initialize items array with
0s

   permutations(cb, nboxes, items, ssf, ritems, "");

   return 0;
}
```

**Step-by-step Execution:**

1. **cb = 1:** We are at the first box. We try all possible items (1 and 2) in the first box.
   - **Item 1:**
     - items[0] = 0, we mark it as used (items[0] = 1).
     - Recursively call permutations(2, 3, [1, 0], 1, 2, "1").
2. **cb = 2:** We are now at the second box. We check all possible items (1 and 2).
   - **Item 1:** (items[0] = 1, already used, so skip).
   - **Item 2:**
     - items[1] = 0, we mark it as used (items[1] = 1).
     - Recursively call permutations(3, 3, [1, 1], 2, 2, "12").
3. **cb = 3:** We are at the third box. We try all possible options:
   - **Item 1 and Item 2:** Both are already used.
   - Add 0 to indicate the third box remains empty.
     - Recursively call permutations(4, 3, [1, 1], 2, 2, "120").
4. **Base case:** cb = 4, output 120.

   **Result:**

   - We print 120 as a valid permutation.

5. **Backtrack:** Unmark the second item (items[1] = 0), and proceed to the next option in cb = 2.
   - **Item 2:**
     - items[1] = 0, mark it as used (items[1] = 1).
     - Recursively call permutations(3, 3, [1, 1], 2, 2, "20").
6. **Base case:** cb = 4, output 120.

Output:-

120