

C++11 code 2.2.12: Solution code for Sub-problem (2-2.e) → [GitLab](#)

```
2  std::vector<Eigen::Triplet<double>> transformCOOmatrix(  
3      const std::vector<Eigen::Triplet<double>> &A) {  
4      std::vector<Eigen::Triplet<double>> A_t{}; // return value  
5  
6      // First step: find the size of the matrix by searching the maximal  
7      // indices. Depends on the assumption that no zero rows/columns  
8      // occur.  
9      int rows_max_idx = 0, cols_max_idx = 0;  
10     for (const Eigen::Triplet<double> &triplet : A) {  
11         rows_max_idx =  
12             (triplet.row() > rows_max_idx) ? triplet.row() : rows_max_idx;  
13         cols_max_idx =  
14             (triplet.col() > cols_max_idx) ? triplet.col() : cols_max_idx;  
15     }  
16     int n_cols = cols_max_idx + 1;  
17  
18     // Make sure we deal with a square matrix  
19     assert(rows_max_idx + 1 == n_cols);  
20     // The matrix size must have even parity  
21     assert(n_cols % 2 == 0);  
22  
23     int N = n_cols; // Size of (square) matrix  
24     int M = N / 2;  // Half the size  
25     // clang-format off  
26     // Distribute entries of "old" matrix to new matrix  
27     for (const Eigen::Triplet<double> &it : A) {  
28         // row and column indices of current A triplet  
29         const int I = it.row() + 1; // k in (2.2.8)–(2.2.11)  
30         const int J = it.col() + 1; // l in (2.2.8)–(2.2.11)  
31  
32         // Distinguish different parities of indices  
33         if (I % 2 == 0 & J % 2 == 0) {  
34             // even, even, (2.2.8)  
35             A_t.emplace_back(I / 2 - 1, J / 2 - 1, it.value());  
36             A_t.emplace_back(I / 2 + M - 1, J / 2 + M - 1, it.value());  
37             A_t.emplace_back(I / 2 + M - 1, J / 2 - 1, -it.value());  
38             A_t.emplace_back(I / 2 - 1, J / 2 + M - 1, -it.value());  
39         } else if (I % 2 != 0 & J % 2 != 0) {  
40             // odd, odd, see (2.2.9)  
41             A_t.emplace_back((I + 1) / 2 - 1, (J + 1) / 2 - 1, it.value());  
42             A_t.emplace_back((I + 1) / 2 + M - 1, (J + 1) / 2 + M - 1, it.value());  
43             A_t.emplace_back((I + 1) / 2 - 1, (J + 1) / 2 + M - 1, it.value());  
44             A_t.emplace_back((I + 1) / 2 + M - 1, (J + 1) / 2 - 1, it.value());  
45         } else if (I % 2 == 0 & J % 2 != 0) {  
46             // even, odd, see (2.2.10)  
47             A_t.emplace_back(I / 2 - 1, (J + 1) / 2 - 1, it.value());  
48             A_t.emplace_back(I / 2 - 1, (J + 1) / 2 + M - 1, it.value());  
49             A_t.emplace_back(I / 2 + M - 1, (J + 1) / 2 + M - 1, -it.value());  
50             A_t.emplace_back(I / 2 + M - 1, (J + 1) / 2 - 1, -it.value());  
51         } else if (I % 2 != 0 & J % 2 == 0) {  
52             // odd, even, see (2.2.11)  
53             A_t.emplace_back((I + 1) / 2 - 1, J / 2 - 1, it.value());  
54             A_t.emplace_back((I + 1) / 2 + M - 1, J / 2 + M - 1, -it.value());  
55             A_t.emplace_back((I + 1) / 2 + M - 1, J / 2 - 1, it.value());  
56             A_t.emplace_back((I + 1) / 2 - 1, J / 2 + M - 1, -it.value());  
57         } else {  
58             assert(false);  
59         }  
60     }  
61 }
```