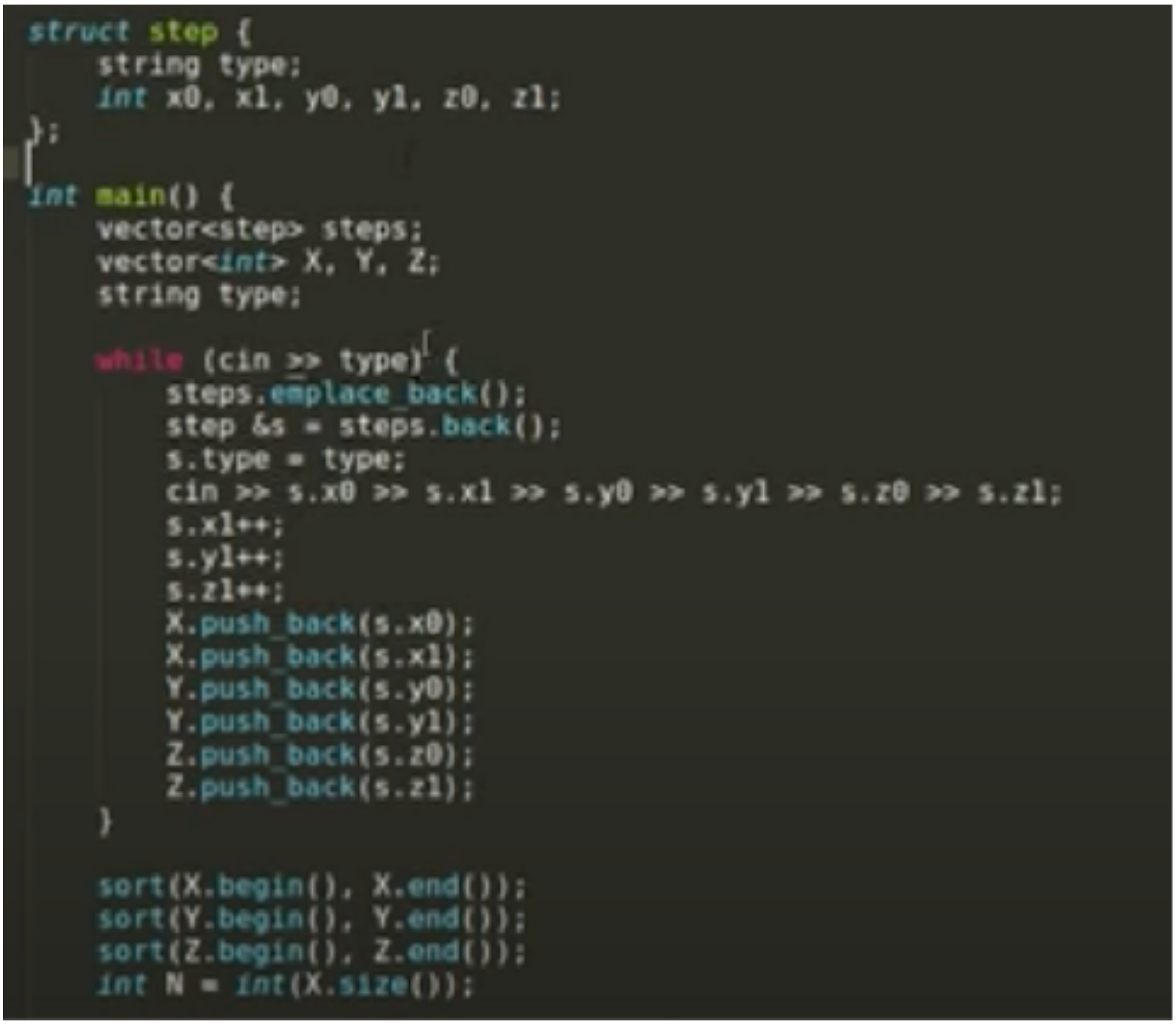
Youtube video "Destroying everyone else in Advent of Code 2021 Day 22" (https://youtu.be/YKpViLcTp64?t=1036)

Neal Wu separates the input cuboids into a vector of "steps" with the on/off flag and the x,y,z ranges for a cuboid, and then also three vectors capital X,Y,Z of of the x,y,z boundaries of the cuboids.



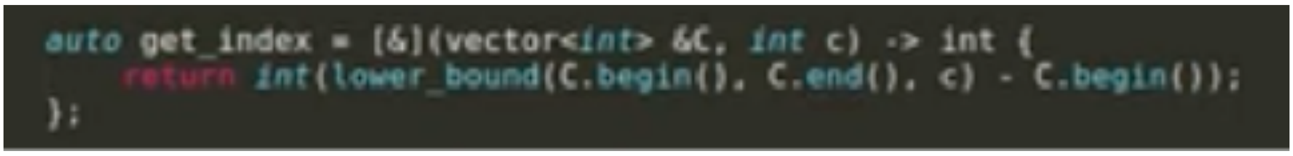
```
struct step {
    string type;
    int x0, x1, y0, y1, z0, z1;
};

int main() {
    vector<step> steps;
    vector<int> X, Y, Z;
    string type;

    while (cin >> type) {
        steps.emplace_back();
        step &s = steps.back();
        s.type = type;
        cin >> s.x0 >> s.x1 >> s.y0 >> s.y1 >> s.z0 >> s.z1;
        s.x1++;
        s.y1++;
        s.z1++;
        X.push_back(s.x0);
        X.push_back(s.x1);
        Y.push_back(s.y0);
        Y.push_back(s.y1);
        Z.push_back(s.z0);
        Z.push_back(s.z1);
    }

    sort(X.begin(), X.end());
    sort(Y.begin(), Y.end());
    sort(Z.begin(), Z.end());
    int N = int(X.size());
```

Then he sorts the X,Y and Z vectors! So now the order of the boundary coordinates in the X,Y,Z no longer line up with the original cuboids! So I suppose for now we don't care?

One way to understand this is that we have separated the x,y and z ranges where "something" happens (on or off range).

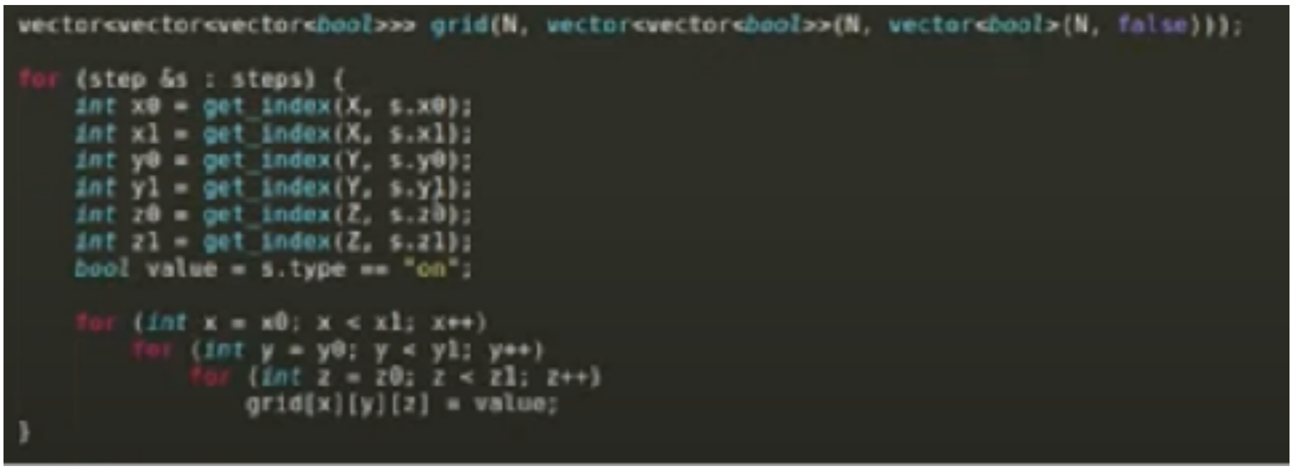
Finally he sets N as the length of the X,Y Z vectors (they are all the same length as every cuboid has a start and stop coordinate for each axes).

He implements a helper function get_index (lets come back to this one below).



```
auto get_index = [&](vector<int> &C, int c) -> int {
    return int(lower_bound(C.begin(), C.end(), c) - C.begin());
};
```

next he "compresses" the cube space into an NxNxN "grid", a space with a dimension to hold only the number of boundaries.



```cpp
vector<vector<vector<bool>>> grid(N, vector<vector<bool>>(N, vector<bool>(N, false)));

for (step &s : steps) {
    int x0 = get_index(X, s.x0);
    int x1 = get_index(X, s.x1);
    int y0 = get_index(Y, s.y0);
    int y1 = get_index(Y, s.y1);
    int z0 = get_index(Z, s.z0);
    int z1 = get_index(Z, s.z1);
    bool value = s.type == "on";

    for (int x = x0; x < x1; x++)
        for (int y = y0; y < y1; y++)
            for (int z = z0; z < z1; z++)
                grid[x][y][z] = value;
}
```

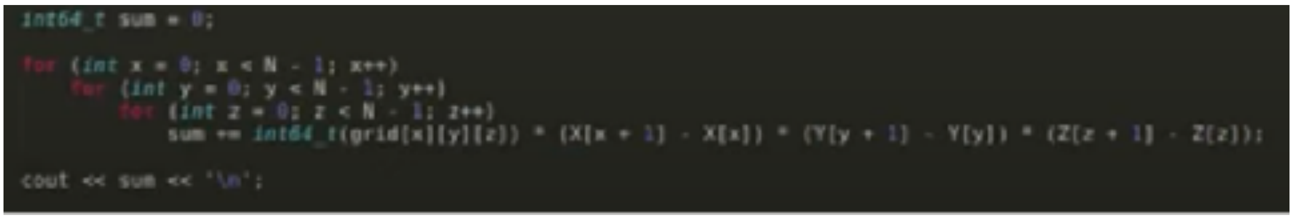
He fills this grid by iterating over the steps in the steps vector.

For each step in the steps vector:

- Create and "index" (call get_index) that is the position in the X,Y and Z vector for each boundary point in the step.
- Then fill all positions in the "grid" between the start and stop index with the on/off state of the range start to stop.

Hm... my head hurts!

Let's see if we can wrap our heads around this by reading the final code.



```cpp
int64_t sum = 0;

for (int x = 0; x < N - 1; x++)
    for (int y = 0; y < N - 1; y++)
        for (int z = 0; z < N - 1; z++)
            sum += int64_t(grid[x][y][z]) * (X[x + 1] - X[x]) * (Y[y + 1] - Y[y]) * (Z[z + 1] - Z[z]);

cout << sum << '\n';
```

Ok, so this looks like summing the volumes of 3D regions that are flagged to be "on".

The grid[x][y][z] is the boolean that is true if the region represented by that grid position is "on".

The size of the region uses the cuboid boundaries from the X,Y,Z vectors to calculate each coordinate dimension. So X[x+1] - X[x] is the actual size of the x-region for the cuboid in question.

I think I understand that bu applying the on/off switching in the order they shall be applied, then the final state in the compressed on-off "grid" will reflect the final state of cubes in the ranges created by the cuboids.

I also watched Jonathan Paulsson "Advent of Code 2021 - Day 22" (https://youtu.be/7gW_h0RTDd8?t=2653) that implemented the same idea :)