- main.cpp

```cpp
#include <iostream>
#include <vector>
#include <string>
#include "maze.h"
#include "robot.h"

enum Face {
    up, right, down, left
};

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    unsigned int row, col, rx, ry;
    unsigned long long step;
    std::cin >> col >> row >> step;
    std::vector<std::string> mp;
    for(size_t i = 0; i < row; ++i) {
        std::string s;
        std::cin >> s;
        for(size_t j = 0; j < col; ++j) {
            if(s[j] == 'O') {
                rx = i;
                ry = j;
                s[j] = '.';
                break;
            }
        }
        mp.push_back(s);
    }
    maze mz(row, col, mp);
    robot bot(rx, ry, Face::up);
    bool repeatFlag = false;
    for(size_t i = 0; i < step; ++i) {
        unsigned int nx, ny;
        bot.getNextPos(nx, ny);
        while(!mz.isCanWalk(nx, ny)) {
            bot.turn(Face::right);
            bot.getNextPos(nx, ny);
        }
        if(!repeatFlag && i > 0) {
            unsigned long long repeatStep = bot.getRepeatPos();
            if(repeatStep > 0) {
                --repeatStep;
                i = step - ((step - repeatStep) % (i - repeatStep)) - 1;
                repeatFlag = true;
                continue;
            }
        }
        bot.goNext();
    }
    bot.getBotPos(rx, ry);
    std::cout << ry << " " << rx << std::endl;
    return 0;
}
```

- maze.h

```cpp
#pragma once
#include <vector>
#include <string>

class maze {
  private:
    const unsigned int row, col;
    const std::vector<std::string> mp;
  public:
    maze(const unsigned int row, const unsigned int col, const std::vector<std::string> mp):
        row(row), col(col), mp(mp) {};
    bool isCanWalk(const int, const int);
};
```

- maze.cpp

```cpp
#include "maze.h"

bool maze::isCanWalk(const int x, const int y) {
    if(x<0 || x>=row || y<0 || y>=col || mp[x][y] == '#') return false;
    return true;
}
```

- robot.h

```cpp
#pragma once
#include <vector>
#include <tuple>

class robot {
  private:
    unsigned int x, y, direction;
    unsigned long long step;
    std::vector<std::tuple<unsigned int, unsigned int, unsigned int>> history;
  public:
    robot(const unsigned int, const unsigned int, const unsigned int);
    static constexpr int d[4][2] = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};
    void getBotPos(unsigned int &, unsigned int &);
    void getNextPos(unsigned int &, unsigned int &);
    void turn(const unsigned int);
    void goNext();
    unsigned long long getRepeatPos();
};
```

- robot.cpp

```
#include <iostream>
#include "robot.h"

robot::robot(const unsigned int ix, const unsigned int iy, const unsigned int idir) {
    x = ix;
    y = iy;
    direction = idir;
    step = 0;
    history.clear();
}

void robot::getBotPos(unsigned int &rx, unsigned int &ry) {
    rx = x;
    ry = y;
}

void robot::getNextPos(unsigned int &nx, unsigned int &ny) {
    nx = x + d[direction][0];
    ny = y + d[direction][1];
}

void robot::turn(const unsigned int td) {
    direction = (direction + td) % 4;
}

void robot::goNext() {
    history.push_back(std::make_tuple(x, y, direction));
    x = x + d[direction][0];
    y = y + d[direction][1];
    ++step;
}

unsigned long long robot::getRepeatPos() {
    for(size_t i = 0; i < history.size(); ++i) {
        if(std::get<0>(history[i]) == x && std::get<1>(history[i]) == y && std::get<2>(history[
            i]) == direction) {
            return i + 1;
        }
    }
    return 0;
}
```