```cpp
// Copyright 2022 Matthew Lorette Anaya

#include "RandWriter.h"

RandWriter::RandWriter(std::string text, int k) {
   rw_txt = text;
    rw_k = k;

    if (rw_txt.length() < static_cast<unsigned int>(rw_k)) {
        throw std::invalid_argument("RandWriter(string text, int k): order k"
        " must be less than or equal to text length.");
    }

    // Table setup
    unsigned int pos = 0;
    for (unsigned int i = 0; i < rw_txt.length(); i++) {
        std::string kgram;
        std::map<char, int> freq_table;

        // kgrams parsing
        for (unsigned int j = i; j < i + rw_k; j++) {
            if (j >= rw_txt.length()) {
                pos = j - rw_txt.length();
            } else {
                pos = j;
            }
            kgram += rw_txt.at(pos);
        }

        // Frequency table setup
        pos++;
        if (pos >= rw_txt.length()) { pos -= rw_txt.length(); }
        freq_table.insert(std::make_pair(rw_txt.at(pos), 0));

        // Mapping
        if (rw_table.count(kgram) == 0) {
            rw_table.insert(std::make_pair(kgram, freq_table));
        }

        rw_table[kgram][rw_txt.at(pos)]++;
    }
}

int RandWriter::orderK() const {
        return rw_k;
}

std::string RandWriter::getText() const {
        return rw_txt;
}

std::map<std::string, std::map<char, int>> RandWriter::get_table() const {
    return rw_table;
}

int RandWriter::freq(std::string kgram) const {
    if (kgram.length() < static_cast<unsigned int>(rw_k)) {
        throw std::runtime_error("freq(string kgram): kgram must be of"
        " length greater than or equal to order k.");
    }
    int count = 0;
    for (unsigned int i = 0; i < rw_txt.length(); i++) {
        unsigned int pos = 0;
        std::string kg;
        // parse input text for kgrams
        for (unsigned int j = i; j < i + rw_k; j++) {
            // get characters for kgrams
            if (j >= rw_txt.length()) {
                pos = j - rw_txt.length();
            } else {
                pos = j;
            }
            kg += rw_txt.at(pos);
        }
        if (kgram == kg) { count++; }
    }
    return count;
}

int RandWriter::freq(std::string kgram, char c) const {
    if (kgram.length() < static_cast<unsigned int>(rw_k)) {
        throw std::runtime_error("freq(string kgram, char c): kgram must be"
        " of length greater than or equal to order k.");
    }
    return rw_table.at(kgram).at(c);
}

char RandWriter::kRand(std::string kgram) const {
    if (kgram.length() < static_cast<unsigned int>(rw_k)) {
        throw std::runtime_error("kRand(string kgram): kgram must be of"
        " length greater than or equal to order k.");
    }
    if (rw_table.count(kgram) == 0) {
        throw std::runtime_error("kRand(string kgram): kgram does not"
        " exist.");
    }
    std::string alphabet;
    for (auto const &var1 : rw_table) {
        if (var1.first == kgram) {
            for (auto const &var2 : var1.second) {
                alphabet += var2.first;
            }
        }
    }
    std::random_device device;
    std::mt19937 mt_rand(device());
    std::uniform_int_distribution<int> distribution(0, alphabet.length()
    - 1);

    return alphabet[distribution(mt_rand)];
}

std::string RandWriter::generate(std::string kgram, int L) const {
    if (kgram.length() < static_cast<unsigned int>(rw_k)) {
        throw std::runtime_error("generate(string kgram, int L): kgram must"
        " be of length greater than or equal to order k.");
    }
    std::string generated = kgram;
    // generate new characters based on kgrams
    for (int i = rw_k; i < L; i++) {
        generated += kRand(generated.substr(i - rw_k, rw_k));
    }
    return generated;
}

std::ostream& operator<<(std::ostream& out, const RandWriter& rw) {
    out << "Markov Model\tOrder: " << rw.rw_k << std::endl;
    out << "kgram:\tfrequency:\tfrqncy of next char:\tprob of next char:" <<
    std::endl;

    for (auto const &var1 : rw.rw_table) {
        // var1.first = kgram
        out << var1.first << "\t";
        out << rw.freq(var1.first) << "\t\t";
        for (auto const &var2 : var1.second) {
            // var2.first = next char
            // var2.second = data
            out << var2.first << ":" << var2.second << " ";
        }
        out << "\t\t\t";
        for (auto const &var2 : var1.second) {
            out << var2.first << ":" << var2.second << "/" <<
            rw.freq(var1.first) << " ";
        }
        out << std::endl;
    }
    return out;
}
```