main.cpp

```cpp
#include<iostream>
#include<vector>
#include <random>

#include <algorithm>


using namespace std;

#define N 200
#define BETA 2
#define T 100000


inline double g(double x) {
    return 1 / (1 + exp(-2 * x * BETA));
}

int stochastic(double x) {

    std::random_device rd;
    std::mt19937 gen(rd());
    std::bernoulli_distribution dis(g(x));

    return dis(gen) ? 1 : -1;
}


auto generatePatterns(int p) {

    vector<vector<int> *> patterns;

    std::random_device rd;
    std::mt19937 gen(rd());
    std::bernoulli_distribution dis(0.5);


    for (int i = 0; i < p; i++) {
        auto vec = new vector<int>;
        for (int j = 0; j < N; j++) {
            vec->push_back(dis(gen) ? 1 : -1);
        }

        patterns.push_back(vec);
    }


    return patterns;

}


class Hopfield {
public:
    double w[N][N] = {0};
```

```
    void train(vector<int> *p) {
        auto pattern = *p;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (i == j)
                    w[i][j] = 0;
                else
                    w[i][j] += ((double) 1 / N) * pattern[i] * pattern[j];
            }
        }

    }


    double calculateOrderParameter(vector<int> *pattern) {
        auto *state = new vector<int>(*pattern);


        double orderParameter = 0.0;

        for (int t = 0; t < T; t++) {

            for (int neuron = 0; neuron < N; neuron++) {


                double res = 0;
                for (int j = 0; j < N; j++) {
                    res += w[neuron][j] * (*state)[j];
                }

                (*state)[neuron] = stochastic(res);
                orderParameter += (*state)[neuron] * (*pattern)[neuron];
            }

            orderParameter /= N;


            if (!(t % 1000)) {
                cout << "Done with " << t << " trials\t" << orderParameter << "\n";
            }

        }

        return orderParameter;

    }
};


int main() {
    auto hopfield = new Hopfield();


    auto answer = 0.0;
    auto times = 0;
    while (times++ <= 100) {
        auto patterns = generatePatterns(40); // 5 (for the first task)

        for (const auto &pattern  : patterns) {
```

```
                hopfield->train(pattern);
            }

            auto temp = hopfield->calculateOrderParameter(patterns[0]);

            cout << "At the end of trial " << times << " we got" << temp;
            answer += temp;
            cout << "\nCurrent average = " << answer / times << endl;
            for (auto pattern: patterns) {
                delete (pattern);
            }
        }

        cout << "Final answer is\n" << (answer / 100) << endl;


        return 0;
    }
```