main.cpp

```cpp
#include <iostream>
#include <random>
#include <math.h>

using namespace std;


double generateRandom(double, double);

double activationFunction(double field);


double tanhDerivative(double val);

int sgn(double x){
    return x>=0?1:-1;
}

#define ETA 0.02
#define N 4


double weights[N] = {0};
double threshold;

int input[16][N] = {
        {- 1, - 1, - 1, - 1},
        {1,   - 1, - 1, - 1},
        {- 1, 1,   - 1, - 1},
        {- 1, - 1, 1,   - 1},
        {- 1, - 1, - 1, 1},
        {1,   1,   - 1, - 1},
        {1,   - 1, 1,   - 1},
        {1,   - 1, - 1, 1},
        {- 1, 1,   1,   - 1},
        {- 1, 1,   - 1, 1},
        {- 1, - 1, 1,   1},
        {1,   1,   1,   - 1},
        {1,   1,   - 1, 1},
        {1,   - 1, 1,   1},
        {- 1, 1,   1,   1},
        {1,   1,   1,   1}
};


int targets[][16] =
        {
                {- 1, - 1, - 1, 1,   - 1, 1,   1,   1,   1,   1,   1,   - 1, 1,   - 1, - 1, - 1},

                {- 1, 1,   1,   - 1, - 1, 1,   - 1, - 1, - 1, - 1, 1,   1,   - 1, 1,   - 1, - 1},
                {1,   - 1, 1,   - 1, - 1, 1,   - 1, - 1, - 1, - 1, - 1, - 1, 1,   - 1, - 1, 1},
                {- 1, 1,   - 1, - 1, - 1, - 1, - 1, - 1, 1,   - 1, - 1, - 1, - 1, - 1, 1,   - 1, - 1},
                {1,   1,   1,   - 1, - 1, 1,   - 1, - 1, 1,   1,   - 1, 1,   1,   - 1, - 1, - 1},
                {1,   1,   - 1, 1,   - 1, - 1, 1,   - 1, 1,   - 1, 1,   1,   - 1, 1,   1,   - 1},
                {1, 1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1}

        };

double getField(const int pattern[N]) {
    double field = 0.0;
    for (int i = 0; i < N; i ++) {
        field += weights[i] * pattern[i];
    }
    return (field - threshold);
}

double predict(const int pattern[N]) {
    return activationFunction(getField(pattern));
}
```

```
int testfunction(int target[16]) {


    int best_acc = 0;

    for (int i = 0; i < N; i ++) {
        weights[i] = generateRandom(- 0.2, 0.2);
    }
    threshold = generateRandom(- 1, 1);

    for (int t = 0; t <= 100000; t ++) {


        double error;

        int mu = (int) (generateRandom(0, 16));

        double field = getField(input[mu]);
        double prediction = activationFunction(field);

        error = tanhDerivative(field) * (target[mu] - prediction);

        for (int j = 0; j < N; j ++) {
            weights[j] += ETA * error * input[mu][j];
        }
        threshold += ETA * error;

        double acc = 0;
        for (int m = 0; m < 16; m ++) {
            auto pred = predict(input[m]);
            if (sgn(pred)==target[m])
                acc ++;
        }
        acc = acc;
        if (acc>best_acc)
            best_acc=acc;
        if (best_acc==16)
            break;
    }


    return best_acc;

}

int main() {

    int best[]={0,0,0,0,0,0,0};

    for (int experiment = 0; experiment<10;experiment++) {
        for (char function = 'A'; function <= 'F'; function ++) {
            int record = testfunction(targets[function - 'A']);
            if (record>best[function-'A']){
                best[function-'A'] = record;
            }
        }
    }


    for (char function = 'A'; function <= 'F'; function ++) {
        cout << "Function " << function << " :"<<100*best[function-'A']/16.0<<"% accuracy"<<endl;

    }

}


double generateRandom(double a, double b) {
    std::random_device rd;
    std::uniform_real_distribution<> uni(a, b);
    return uni(rd);
}

double activationFunction(double field) {
```

```
    return tanh(field);
}

double tanhDerivative(double val) {
    return (1.0 - pow(tanh(val), 2.0));
}
```