#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <algorithm>

#include <future>

using namespace std;

using namespace chrono;

constexpr size\_t MIN\_LEN = 1;

constexpr size\_t MAX\_LEN = 10;

constexpr size\_t MAX\_VALUE = 100;

int thread\_count;

class Timer

{

private:

steady\_clock::time\_point start\_time;

public:

Timer() : start\_time(steady\_clock::now()) {}

~Timer()

{

duration<double> elapsed\_seconds = steady\_clock::now() - start\_time;

cout << "elapsed time: " << elapsed\_seconds.count() << "sec\n";

}

};

/// <summary>

/// Генерирует вектор строк

/// </summary>

/// <returns>вектор строк</returns>

vector<string> GenerateVector()

{

vector<string> v;

v.reserve(MIN\_LEN + (rand() % (MAX\_LEN - MIN\_LEN)));

for (size\_t i = 0; i < v.capacity(); i++)

{

stringstream ss;

ss << rand() % MAX\_VALUE;

v.push\_back(ss.str());

}

return v;

}

vector<string> InputVector()

{

vector<string> v;

string str;

getline(cin, str, '\n');

stringstream ss(str);

while (ss >> str)

{

v.push\_back(str);

}

return v;

}

/// <summary>

/// Выводит вектор в поток

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="os">выходной поток</param>

/// <param name="v">вектор</param>

/// <returns></returns>

template<typename T>

ostream& operator<<(ostream& os, const vector<T>& v)

{

for (const auto& i : v)

{

os << "(" << i << ") ";

}

return os;

}

/// <summary>

/// вычисляет прямое произведение для части левого множества и полного правого

/// </summary>

/// <param name="chunk">часть левого множества</param>

/// <param name="src">второе множество</param>

/// <returns>прямое произведение</returns>

vector<string> startThread(const vector<string>& chunk, const vector<string>& src)

{

vector<string> strs;

for (const auto& item\_chunk : chunk)

{

for (const auto& B\_item : src)

{

stringstream ss;

ss << item\_chunk << ", " << B\_item;

strs.push\_back(ss.str());

}

}

return strs;

}

/// <summary>

/// параллельно вычисляет прямое произведение

/// </summary>

/// <param name="A">первое множество</param>

/// <param name="B">второе множество</param>

/// <returns>прямое произведение</returns>

vector<string> Multiply(const vector<string>& A, const vector<string>& B)

{

vector<future<vector<string>>> threads; // вектор потоков

vector<string> result;

const int shift = A.size() / (thread\_count - 1); // смещение (размер chunk)

// start thread

for (size\_t i = 0; i < thread\_count - 1; i++)

{

vector<string> A\_chunk(A.begin() + min(i \* shift, A.size()), A.begin() + min((i + 1) \* shift, A.size()));

threads.push\_back(async(startThread, move(A\_chunk), B));

}

// оставшийся не полный кусок

vector<string> last\_chunk(A.begin() + (thread\_count - 1) \* shift, A.end());

threads.push\_back(async(startThread, move(last\_chunk), B));

// join'им thread'ы

for (auto& thread : threads)

{

auto th = thread.get();

// todo: оптимизировать

for (auto& i : th)

{

result.push\_back(i);

}

}

return result;

}

int main()

{

srand(time(NULL));

cout << "Count of threads (>1): ";

cin >> thread\_count;

// по усл. - многопоточное приложение

if (thread\_count < 2)

{

cerr << "ERROR! thread\_count < " << thread\_count;

exit(-1);

}

// генерируем массивы

//vector<string> A(move(GenerateVector()));

//vector<string> B(move(GenerateVector()));

//vector<string> C(move(GenerateVector()));

//vector<string> D(move(GenerateVector()));

cout << "[A] = ";

cin.ignore(256, '\n');

vector<string> A = InputVector();

cout << "[B] = ";

vector<string> B = InputVector();

cout << "[C] = ";

vector<string> C = InputVector();

cout << "[D] = ";

vector<string> D = InputVector();

// выполняем распараллеленное прямое произведение для каждого множества

// можно было заморочиться и сделать цикл, но тут не много множеств, поэтому оставлю так :)

auto ab = Multiply(A, B);

cout << '\n' << "A \* B = " << '\n' << ab << '\n';

auto abc = Multiply(ab, C);

cout << '\n' << "(A\*B)\*C = " << '\n' << abc << '\n';

auto abcd = Multiply(abc, D);

cout << '\n' << "((A\*B)\*C)\*D = " << '\n' << abcd << '\n';

}