Отчет по лабораторной работе №1

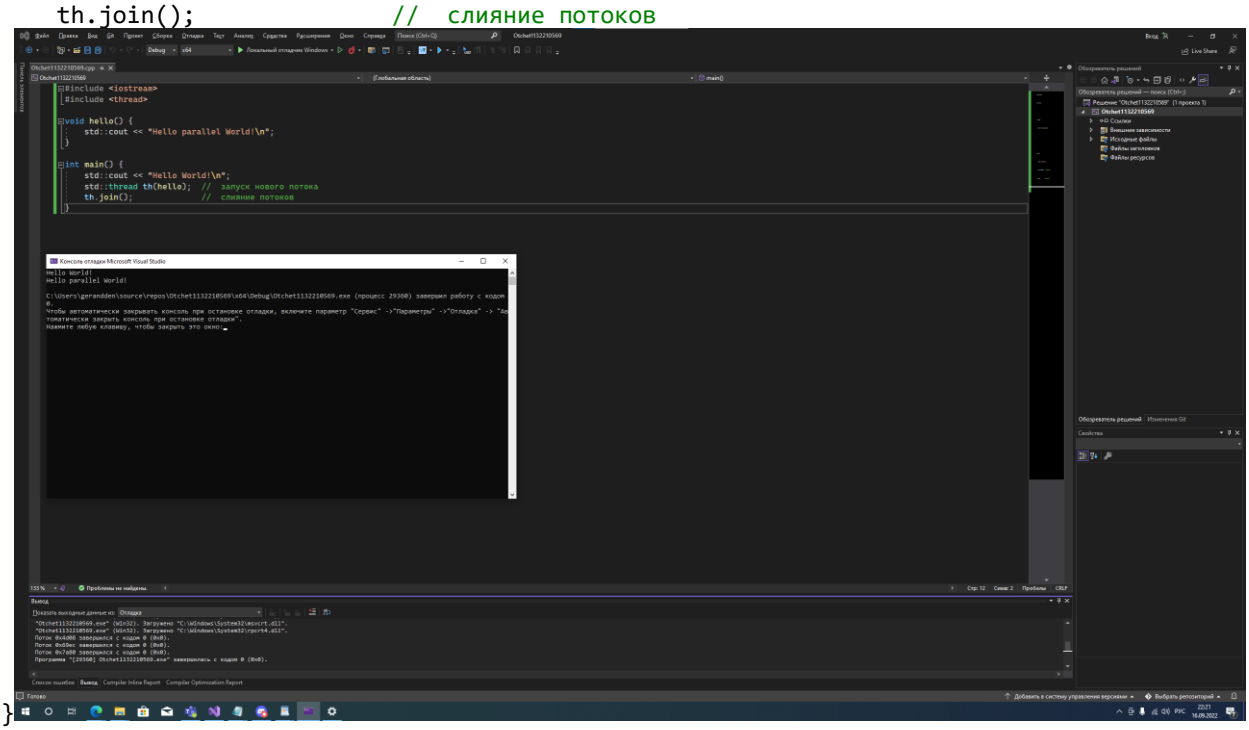
Выполнил Герасимов АД, ИУСбд-01-20, 1132210569

Example 1.1

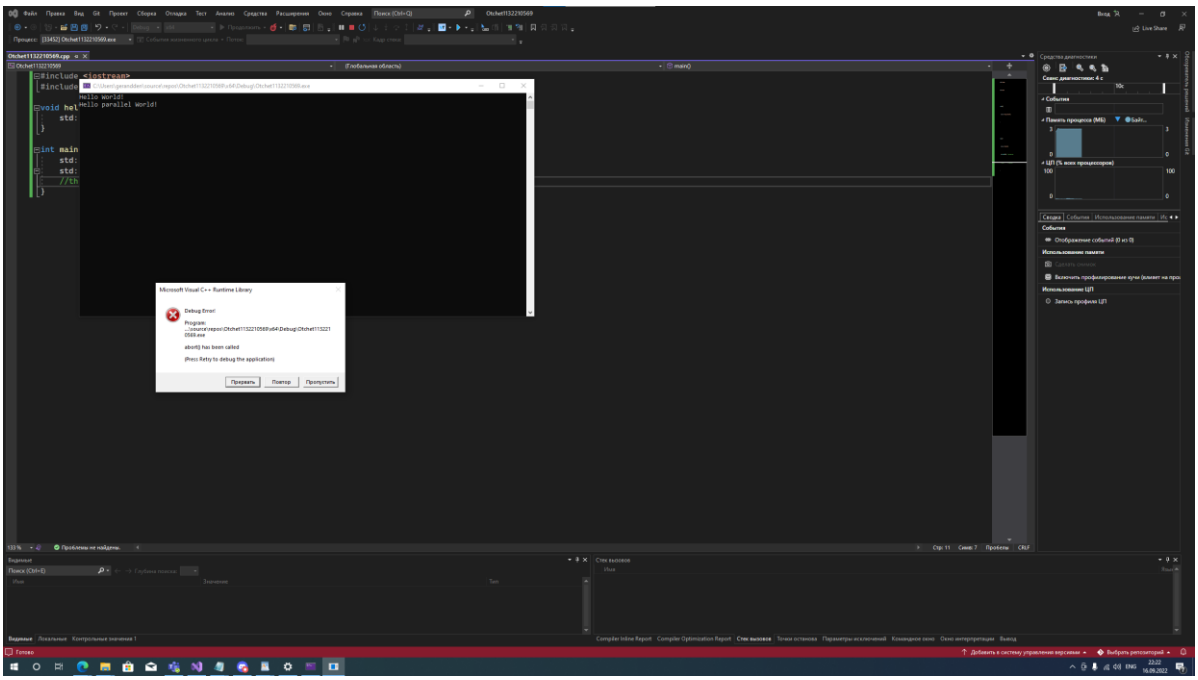
1. Создайте для вывода преветствия отдельный поток, для этого добавьте в программу:

```
#include <thread>  //  https://www.cplusplus.com/reference/thread/thread/

void hello(){
    std::cout<<"Hello parallel World!\n";
}

и в функцию main:

std::thread th(hello);  //  запуск нового потока
th.join();              //  слияние потоков
```

```
#include <iostream>
#include <thread>

void hello() {
    std::cout << "Hello parallel World!\n";
}

int main() {
    std::cout << "Hello World!\n";
    std::thread th(hello);  //  запуск нового потока
    th.join();              //  слияние потоков
}
```



2. Уберите строку th.join(); и попробуйте запустить программу.
Каккие варианты работы программы мы можем увидеть? Объясните результат.



если убрать строчку join, то у нас получится бесконечно открытый поток, который будет требовать ресурсов и работу, которой фактичекски нет.

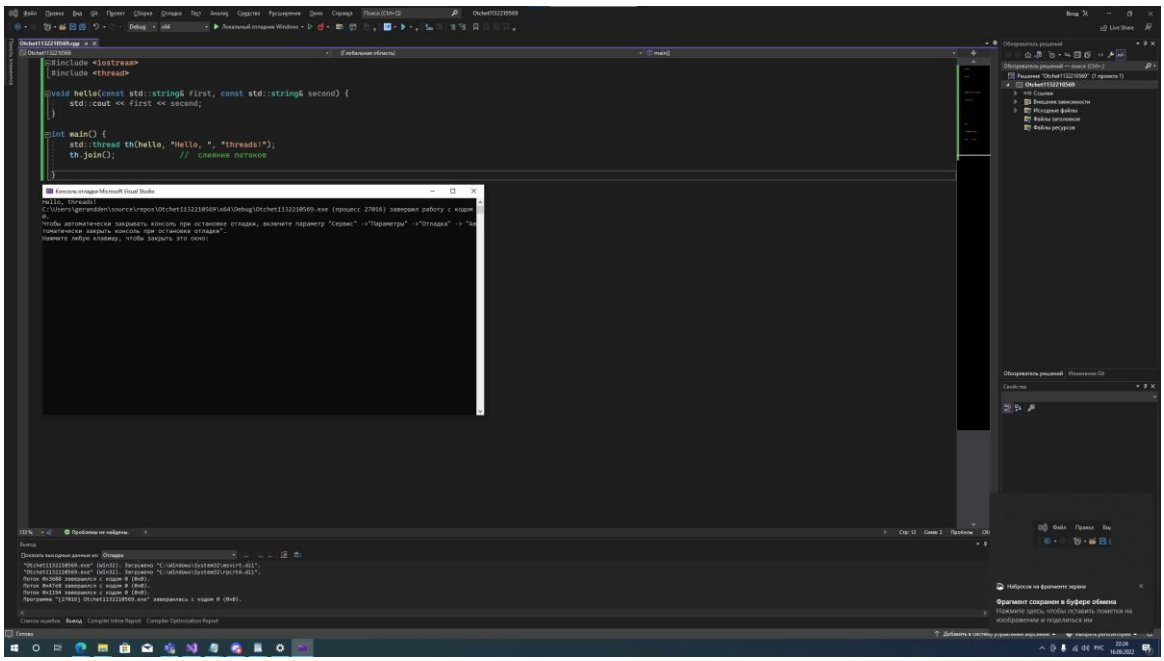
3. Передайте в поток данные для вывода на экран.
Для этого преобразуйте функцию вывода:

```
void hello(const std::string& first, const std::string& second) {
    std::cout << first << second;
}
```

и вызов потока

```
std::thread th(hello, "Hello, ", "threads!");
```

```
#include <iostream>
#include <thread>

void hello(const std::string& first, const std::string& second) {
    std::cout << first << second;
}

int main() {
    std::thread th(hello, "Hello, ", "threads!");
    th.join();              //  слияние потоков
}
```



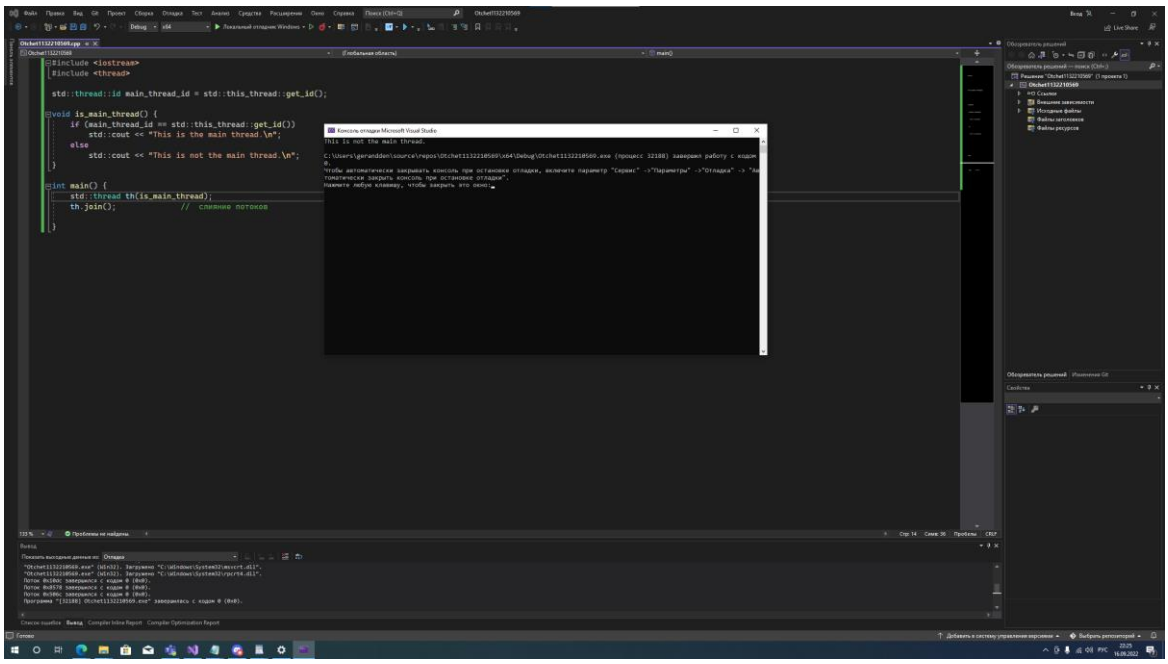
4. При работе с потоками бывает важно знать в каком именно потоке мы находимся. Это можно сделать с помощью функции-метода get_id класса thread (https://www.cplusplus.com/reference/thread/thread/get_id/)

Дополните программу следующим кодом (разберите его назначение):

```
std::thread::id main_thread_id = std::this_thread::get_id();

void is_main_thread() {
  if ( main_thread_id == std::this_thread::get_id() )
    std::cout << "This is the main thread.\n";
  else
    std::cout << "This is not the main thread.\n";
}
```

Используйте функцию is_main_thread в main и запущенном потоке.

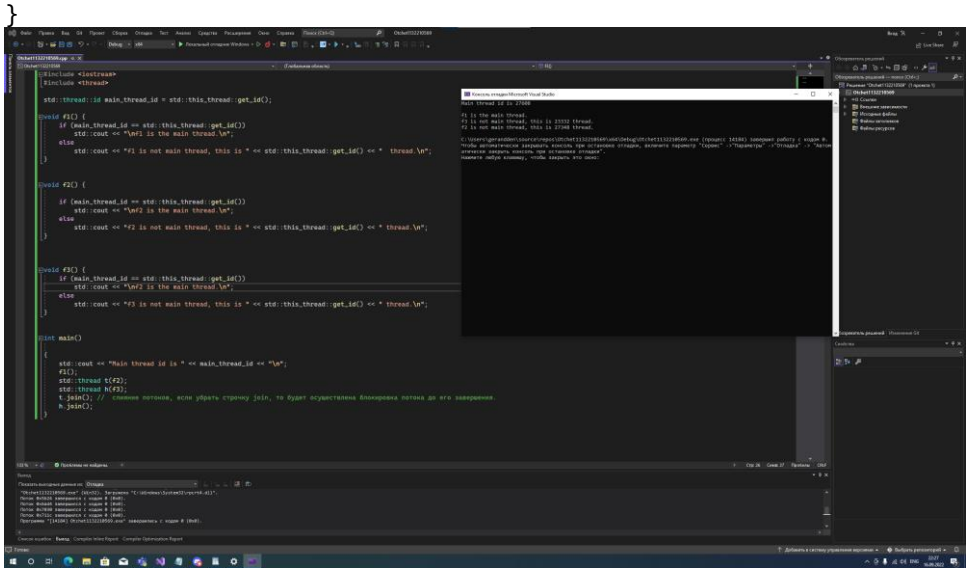
```
#include <iostream>
#include <thread>

std::thread::id main_thread_id = std::this_thread::get_id();

void is_main_thread() {
    if (main_thread_id == std::this_thread::get_id())
        std::cout << "This is the main thread.\n";
    else
        std::cout << "This is not the main thread.\n";
}

int main() {
    std::thread th(is_main_thread);
    th.join();              //  слияние потоков
}
```



5. Перепишите функцию приветствия потоков таким образом, чтобы в случае главного потока выводилось:
Hello, main threads!
а в любом другом потоке вместо слова main стоял идетификатор потока.
Запустите 3 потока в прграмме.

```
#include <iostream>
#include <thread>

std::thread::id main_thread_id = std::this_thread::get_id();

void aboba() {
    if (main_thread_id == std::this_thread::get_id())
        std::cout << "\nAboba is the main thread.\n";
    else
        std::cout << "Aboba is not main thread, this is " << std::this_thread::get_id()
<< "  thread.\n";
}

void abyba() {

    if (main_thread_id == std::this_thread::get_id())
        std::cout << "\nAbyba is the main thread.\n";
    else
        std::cout << "Abyba is not main thread, this is " << std::this_thread::get_id()
<< " thread.\n";
}

void aboby() {
    if (main_thread_id == std::this_thread::get_id())
        std::cout << "\nAboby is the main thread.\n";
    else
        std::cout << "Aboby is not main thread, this is " << std::this_thread::get_id()
<< " thread.\n";
}


int main()
{
    std::cout << "Main thread id is " << main_thread_id <<"\n";
```

```
    aboba();
    std::thread t(abyba);
    std::thread h(aboby);
    t.join(); //  слияние потоков, если убрать строчку join, то будет осуществлена блокировка потока до его завершения.
    h.join();
}
```

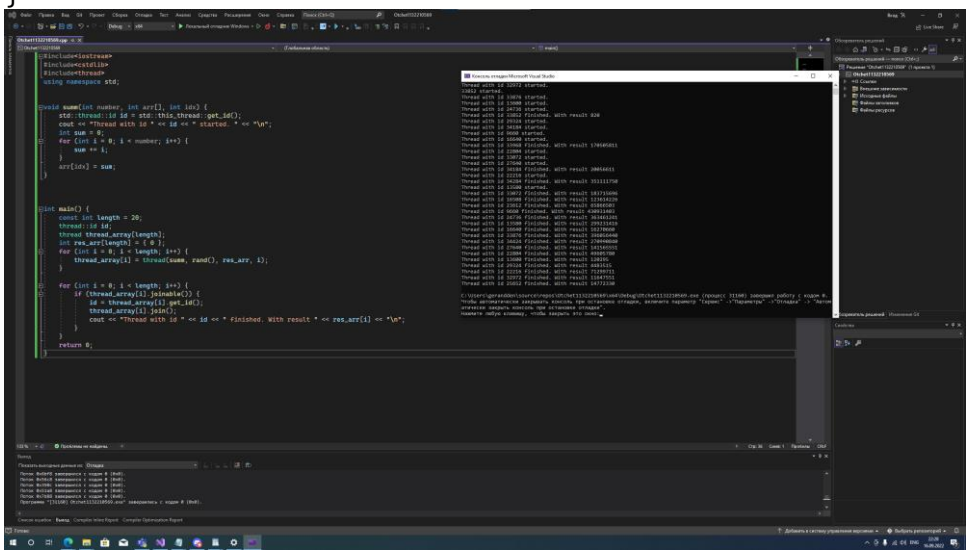


Example 1.2

1. Обратите внимание на ошибки (непоследовательность) при выводе сообщения о старте потока.
Чем это вызвано? Почему этого не происходит при выводе результата работы потока? Как можно решить эту проблему?
(Подсказка: каким ресурсом хотят воспользоваться потоки и как организовать эксклюзивный доступ к нему.)

```
#include<iostream>
#include<cstdlib>
#include<thread>
using namespace std;


void summ(int number, int arr[], int idx) {

    std::thread::id id = std::this_thread::get_id();
    cout << "Thread with id " << id << " started. " << "\n";
    int sum = 0;
    for (int i = 0; i < number; i++) {
        sum += i;
    }
    arr[idx] = sum;
}

int main() {
    const int length = 20;
    thread::id id;
    thread thread_array[length];
    int res_arr[length] = { 0 };
```

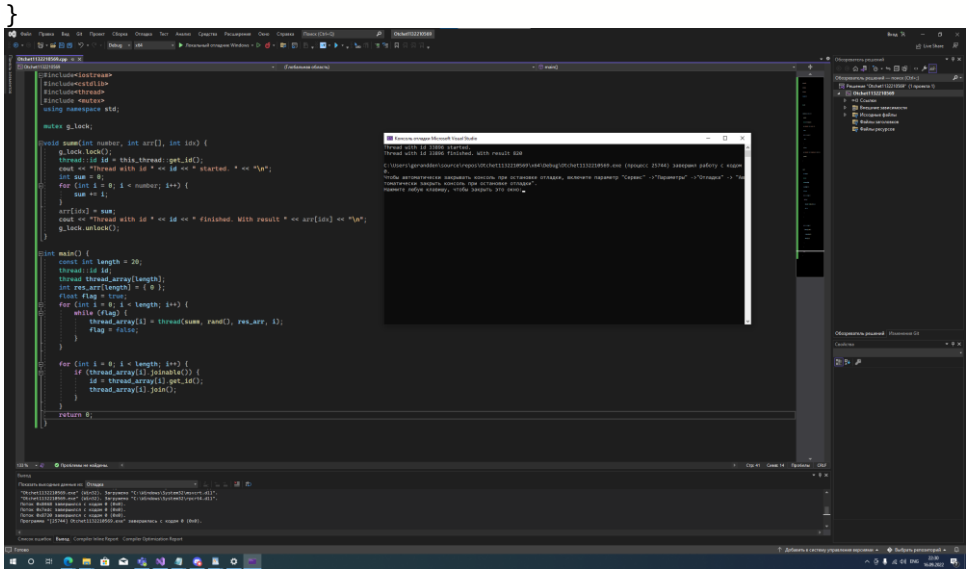
```
    for (int i = 0; i < length; i++) {
        thread_array[i] = thread(summ, rand(), res_arr, i);
    }
    for (int i = 0; i < length; i++) {
        if (thread_array[i].joinable()) {
            id = thread_array[i].get_id();
            thread_array[i].join();
            cout << "Thread with id " << id << " finished. With result " <<
res_arr[i] << "\n";
        }
    }
    return 0;
}
```



2. Доработайте программу таким образом, что расчёты в потоках останавливаются после того, как хотя бы один из них
завершился. После этого цикл прерывается, а запись в arr не производится. Проверьте правильность работы разработанной программы замером времени работы потоков (или временем их завершения).

```
#include<iostream>
#include<cstdlib>
#include<thread>
#include <mutex>
using namespace std;

mutex g_lock;

void summ(int number, int arr[], int idx) {
    g_lock.lock();
    thread::id id = this_thread::get_id();
    cout << "Thread with id " << id << " started. " << "\n";
    int sum = 0;
    for (int i = 0; i < number; i++) {
        sum += i;
    }
    arr[idx] = sum;
    cout << "Thread with id " << id << " finished. With result " << arr[idx] << "\n";
    g_lock.unlock();
}
```

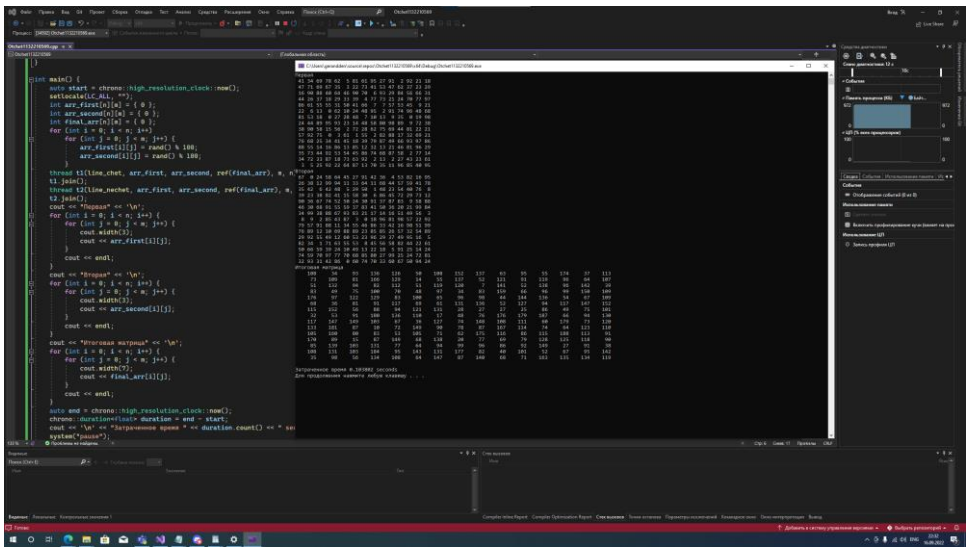
```
int main() {
    const int length = 20;
    thread::id id;
    thread thread_array[length];
    int res_arr[length] = { 0 };
    float flag = true;
    for (int i = 0; i < length; i++) {
        while (flag) {
            thread_array[i] = thread(summ, rand(), res_arr, i);
            flag = false;
        }
    }

    for (int i = 0; i < length; i++) {
        if (thread_array[i].joinable()) {
            id = thread_array[i].get_id();
            thread_array[i].join();
        }
    }
    return 0;
}
```



3.1. Напишите программу производящую паралельное суммирование двух матриц. Результат записывается в матрице продукт и выводится на экран.

```
#include <iostream>
#include <thread>
#include <chrono>
using namespace std;
const int n = 10;
const int m = 10;

void line_nechet(int(*numbers1)[m], int(*numbers2)[m], int(*numbersfinal)[m], int
maxline, int maxstolb) {
    for (int i = 0; i < maxstolb; i += 2) {
        for (int j = 0; j < maxline; j++) {
            numbersfinal[i][j] = numbers1[i][j] + numbers2[i][j];
        }
    }
}

void line_chet(int(*numbers1)[m], int(*numbers2)[m], int(*numbersfinal)[m], int maxline,
int maxstolb) {
    for (int i = 1; i < maxstolb; i += 2) {
        for (int j = 0; j < maxline; j++) {
            numbersfinal[i][j] = numbers1[i][j] + numbers2[i][j];
```

```
        }
    }
}


int main() {
    auto start = chrono::high_resolution_clock::now();
    setlocale(LC_ALL, "");

    int arr_first[n][m] = { 0 };
    int arr_second[n][m] = { 0 };
    int final_arr[n][m] = { 0 };

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++) {
            arr_first[i][j] = rand() % 100;
            arr_second[i][j] = rand() % 100;
        }

    thread t1(line_chet, arr_first, arr_second, ref(final_arr), m, n);
    t1.join();
    thread t2(line_nechet, arr_first, arr_second, ref(final_arr), m, n);

    t2.join();
    cout << "Первая" << '\n';
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout.width(3);
            cout << arr_first[i][j];
        }
        cout << endl;
    }
    cout << "Вторая" << '\n';
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout.width(3);
            cout << arr_second[i][j];
        }
        cout << endl;
    }
    cout << "Итоговая матрица" << '\n';
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout.width(7);
            cout << final_arr[i][j];
        }
        cout << endl;
    }
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<float> duration = end - start;
    cout << '\n' << "Затраченное время " << duration.count() << " seconds" << '\n';
    system("pause");
    return 0;
}
```



3.2. Напишите программу производящую параллельное умножение двух матриц. Результат записывается в матрице продукт и выводится на экран.

```
#include <iostream>
#include <thread>
#include <chrono>
using namespace std;
const int n = 10;
const int m = 10;

void line_nechet(int(*numbers1)[m], int(*numbers2)[m], int(*numbersfinal)[m], int
maxline, int maxstolb) {
    for (int i = 0; i < maxstolb; i += 2) {
        for (int j = 0; j < maxline; j++) {
            numbersfinal[i][j] = numbers1[i][j] * numbers2[i][j];
        }
    }
}

void line_chet(int(*numbers1)[m], int(*numbers2)[m], int(*numbersfinal)[m], int maxline,
int maxstolb) {
    for (int i = 1; i < maxstolb; i += 2) {
        for (int j = 0; j < maxline; j++) {
            numbersfinal[i][j] = numbers1[i][j] * numbers2[i][j];
        }
    }
}


int main() {
    auto start = chrono::high_resolution_clock::now();
    setlocale(LC_ALL, "");

    int arr_first[n][m] = { 0 };
    int arr_second[n][m] = { 0 };
    int final_arr[n][m] = { 0 };

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++) {
            arr_first[i][j] = rand() % 100;
            arr_second[i][j] = rand() % 100;
        }

    thread t1(line_chet, arr_first, arr_second, ref(final_arr), m, n);
    t1.join();
    thread t2(line_nechet, arr_first, arr_second, ref(final_arr), m, n);
```

```
    t2.join();
    cout << "Первая" << '\n';
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout.width(3);
            cout << arr_first[i][j];
        }
        cout << endl;
    }
    cout << "Вторая" << '\n';
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout.width(3);
            cout << arr_second[i][j];
        }
        cout << endl;
    }
    cout << "Итоговая матрица" << '\n';
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout.width(7);
            cout << final_arr[i][j];
        }
        cout << endl;
    }
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<float> duration = end - start;
    cout << '\n' << "Затраченное время " << duration.count() << " seconds" << '\n';
    system("pause");
    return 0;
}
```