ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
“ВЫСШАЯ ШКОЛА ЭКОНОМИКИ”»

ДОМАШНЕЕ ЗАДАНИЕ №4

Работа студента 2 курса бакалавриата группы БПИ-195

по предмету «Архитектура вычислительных систем»

Выполнил:

Карабаш Радимир Сейранович,

БПИ195-1

Преподаватель:

Доктор технических наук,

Профессор

Легалов А. И

Москва 2020

Задание:

Определить множество индексов i, для которых (A[i] - B[i]) или (A[i] + B[i]) являются простыми числами. Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков является входным параметром.

Решение:

Так как в программе используется метод, внутри которого находится цикл для проверки выполнения условия, и все потоки принадлежат данному методу для решения общей задачи, было принято решение использовать **итеративный параллелизм** в качестве модели построения приложения.

При запуске программы пользователь видит приветствующее сообщение с информацией об исполнителе и задании, а затем программа просит ввести пользоваетеля длину двух массивов. После получения информации о длине массивов программа просит пользователя ввести количество потоков для выполнения задачи. Два массива генерируются с помощью генератора псевдослучайных чисел, и логируются в файл input.txt для дальнейшей проверки корректности алгоритма.

Далее вызывается метод **get_indexes**, внутри которого происходит распараллеливание программы с помощью директив OpenMP, который принимает на вход соответствующие два массива, и ссылку на массив с индексами простых чисел. Далее после отработки указанного выше метода список индексов сортируется, логирутся в файл output.txt для дальнейшей проверки корректности алгоритма, и показывается пользователю.

Вывод данных осуществляется в консоль.

Для проверки правильности работы алгоритма реализована логирование входных и выходных данных! (файлы input.txt и output.txt).

Входным параметром для количества потоков является целое число в диапазоне от 1 до 10 включительно.

Описание методов:

- **vector<int> generate_array(int size) –** генерация массива длиной size
- **void is_prime(int n, bool& flag) –** проверка числа n на простое
- **bool try_parse(string n) –** возможно ли привести строку n в целочисленный тип
- **int introduce(int& thr_count) –** получение входных параметров и вывод информативного сообщения
- **void log(vector<int> a, vector<int> b) –** логирование входных данных
- **void log(vector<int> a) –** логирование выходных данных
- **void get_indexes(vector<int> a, vector<int> b, vector<int>& indexes) –** получение индексов простых чисел

Текст программы приведен ниже:

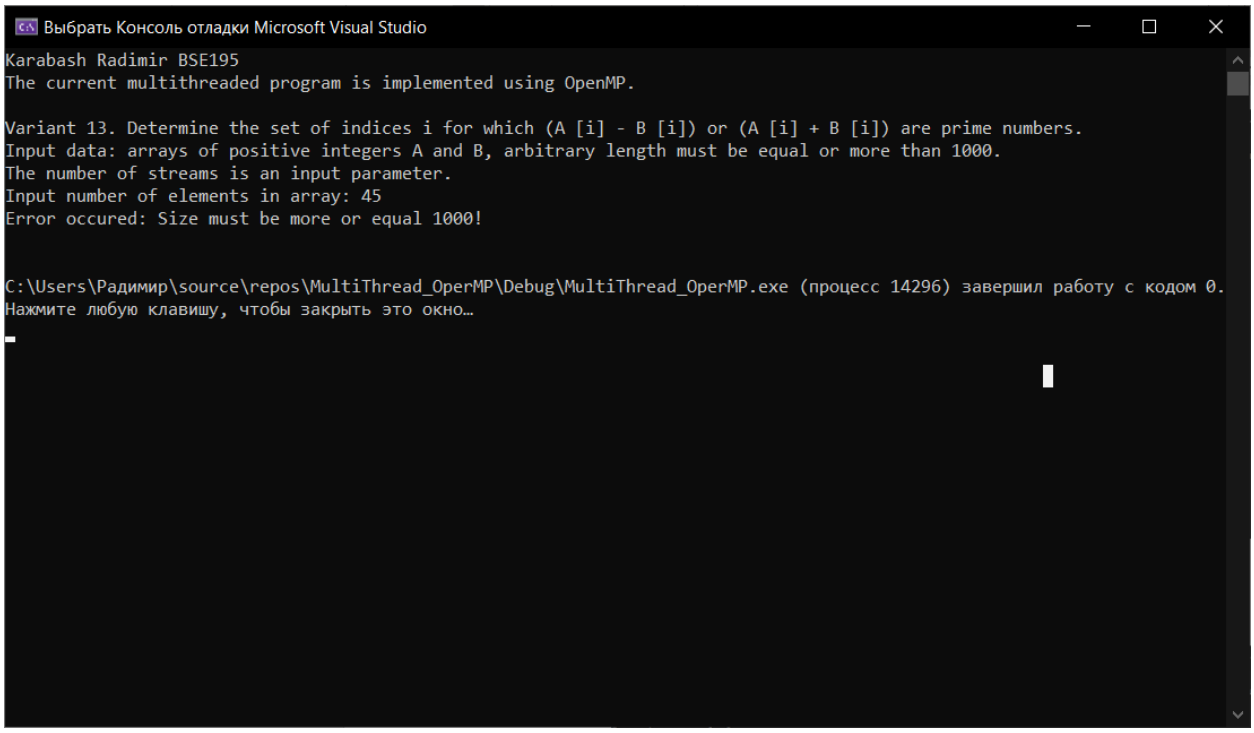
```
// c++ MultiThread_OperMP.cpp -fopenmp
#include <iostream>
#include<vector>
#include<string>
#include<omp.h>
#include<fstream>
#include<algorithm>

using namespace std;
//Генерация массива
vector<int> generate_array(int size) {
    vector<int> v1(size);
    for (size_t i = 0; i < v1.size(); i++)
    {
        v1[i] = 1 + rand();
    }
    return v1;
}
// Проверка на простое число
void is_prime(int n, bool& flag) {
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            flag = false;
            return;
        }
    }
    flag = true;
}
// Возможно ли записать значение в int
bool try_parse(string n) {
    try
```

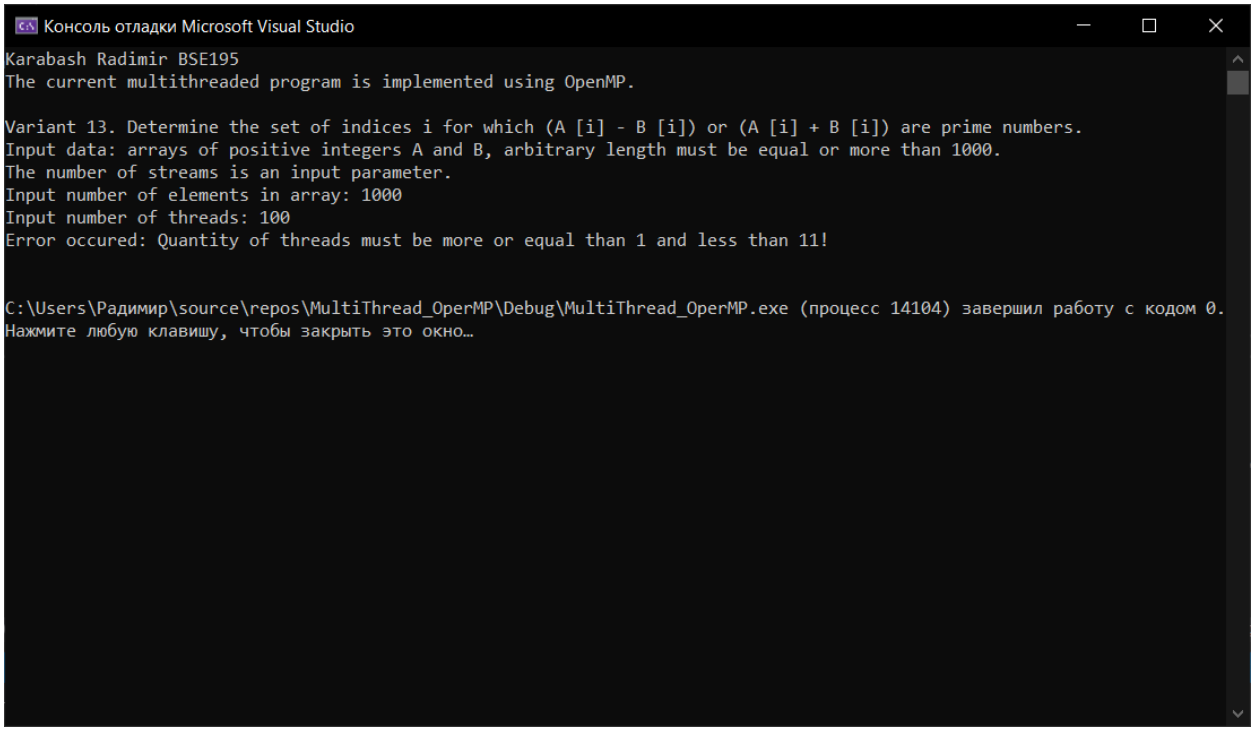
```
    {
        int a = stoi(n);
        return true;
    }
    catch (const std::exception&)
    {
        return false;
    }
}
// Ознакомительные сообщения и получение данных от пользователя
int  introduce(int& thr_count) {
    std::cout << "Karabash Radimir BSE195\nThe current multithreaded program
is implemented using OpenMP.\n\n";
    std::cout << "Variant 13. Determine the set of indices i for which (A [i]
- B [i]) or (A [i] + B [i]) are prime numbers.\nInput data: arrays of
positive integers A and B, arbitrary length must be equal or more than 1000.
\nThe number of streams is an input parameter. \n";
    cout << "Input number of elements in array: ";
    string n;
    cin >> n;
    int size;
    if (try_parse(n))
    {
        size = stoi(n);
        if (size < 1000)
        {
            throw new exception("Size must be more or equal 1000!\n");
        }
    }
    else
    {
        throw new exception("Invalid size of array!\n");
    }
    cout << "Input number of threads: ";
    n = "";
    cin >> n;
    if (try_parse(n))
    {
        thr_count = stoi(n);
        if (thr_count < 1 || thr_count > 10)
        {
            throw new exception("Quantity of threads must be more or equal
than 1 and less than 11!\n");
        }
    }
    else
    {
        throw new exception("Invalid quantity of threads!\n");
    }
    return size;
}
// Логирование входных и выходных данных
void log(vector<int> a, vector<int> b) {
    fstream file("../input.txt", fstream::out);
    file << "Array A: ";
    for (size_t i = 0; i < a.size(); i++)
    {
        file << to_string(a[i]) + " ";
    }
    file << "\n\nArray B: ";
    for (size_t i = 0; i < b.size(); i++)
    {
        file << to_string(b[i]) + " ";
    }
```

```
}
void log(vector<int> a) {
    fstream file("../output.txt", fstream::out);
    file << "Indexes: ";
    for (size_t i = 0; i < a.size(); i++)
    {
        file << to_string(a[i]) + " ";
    }
}
// Получение индексов простых чисел
void get_indexes(vector<int> a, vector<int> b, vector<int>& indexes) {
    bool left, right;
    #pragma omp parallel
    {
        #pragma omp for
        for (int i = 0; i < a.size(); i++)
        {
            is_prime(abs(a[i] - b[i]), left);
            is_prime(abs(a[i] + b[i]), right);
            if (left || right)
            {
            #pragma omp critical
                {
                    indexes.push_back(i);
                    //cout << "Thread(" << omp_get_thread_num() << ") Index:"
<< to_string(i) << endl;
                }
            }
        }
    }
}
int main()
{
    try {
        int thr_count;
        int size = introduce(thr_count);
        vector<int> a = generate_array(size);
        vector<int> b = generate_array(size);
        log(a, b);
        vector<int> indexes;
        omp_set_num_threads(thr_count);
        get_indexes(a, b, indexes);
        sort(indexes.begin(), indexes.end());
        log(indexes);
        for (size_t i = 0; i < indexes.size(); i++)
        {
            cout << indexes[i] << endl;
        }
        cout << "Logging of input and output data is implemented to check the
correctness of the algorithm! (files input.txt and output.txt)";
    }
    catch (exception* e) {
        cout << "Error occured: " << (*e).what() << endl;
    }
}
```

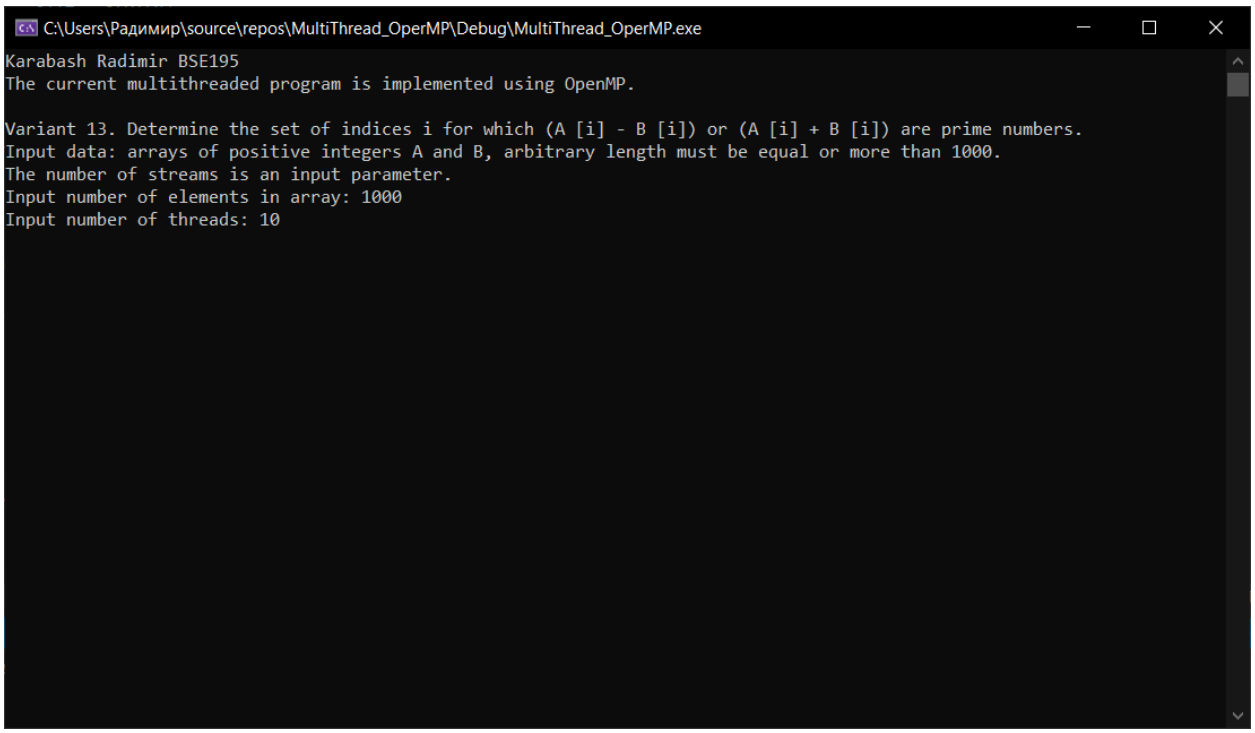
Результаты разработки:



TEST 1

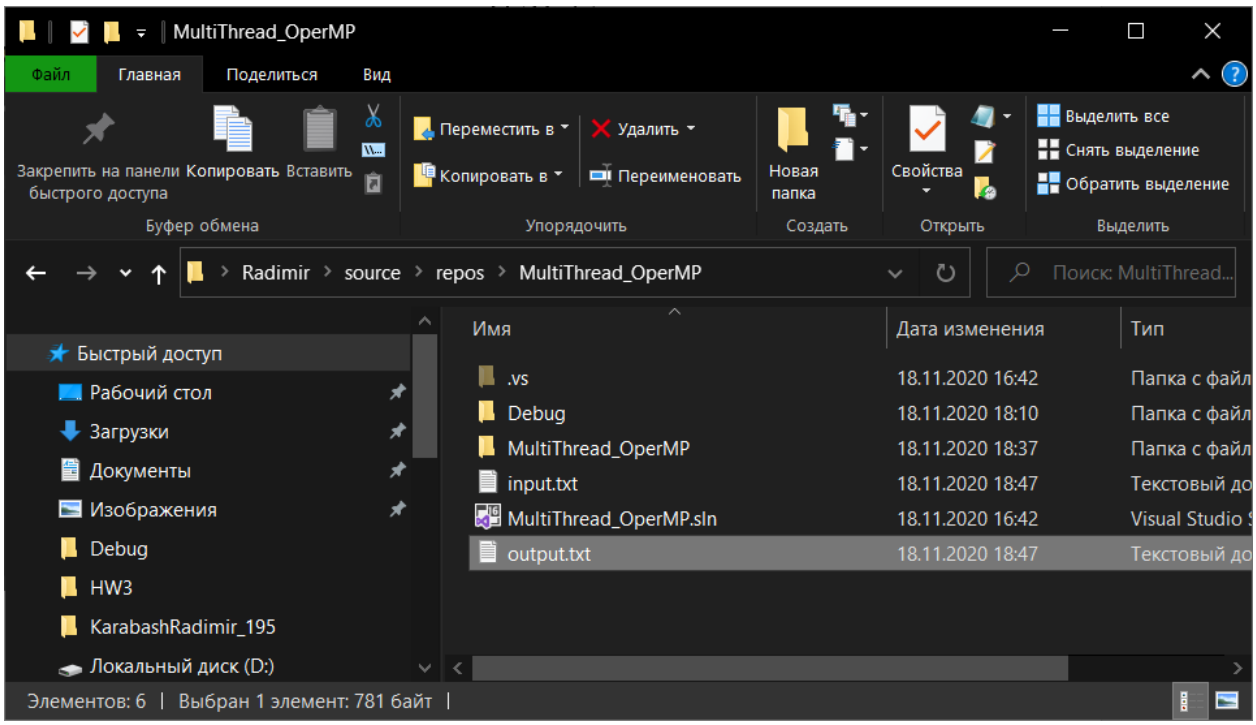


TEST 2

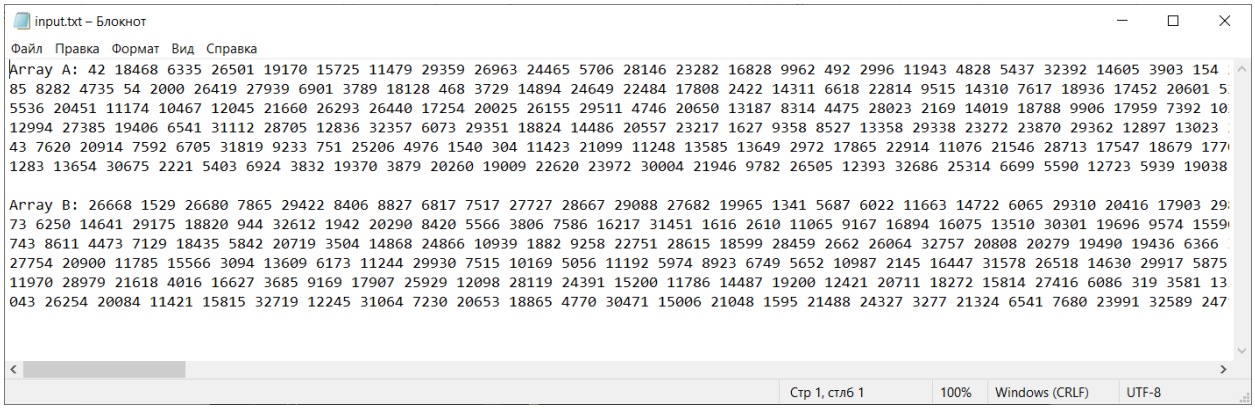


TEST 3.1

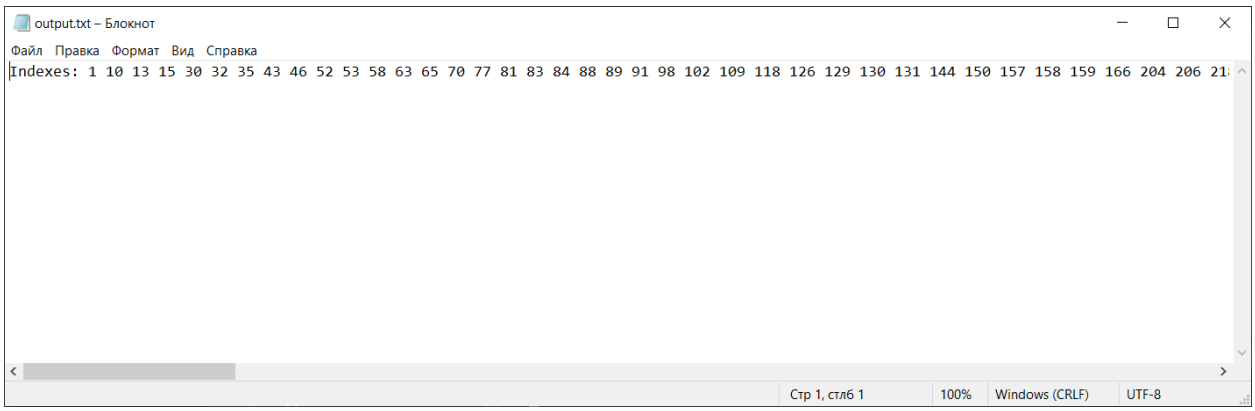
TEST 3.2



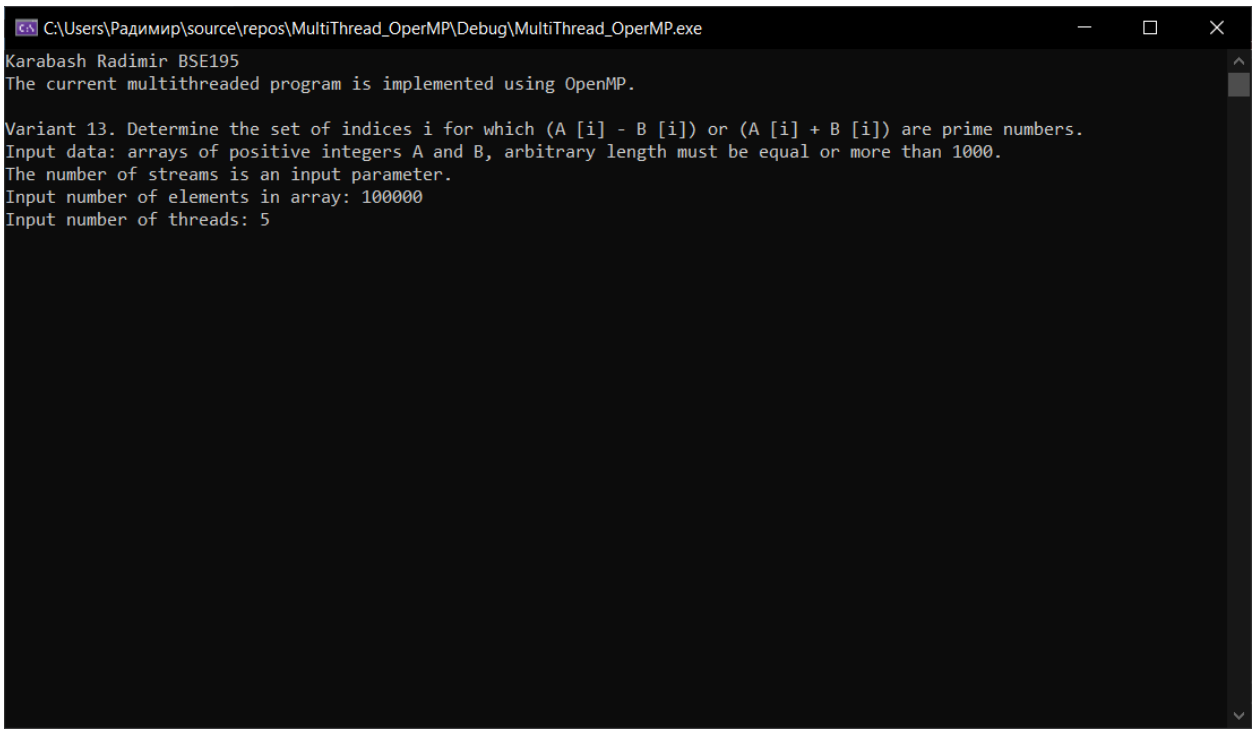
TEST 3.3



TEST 3.4

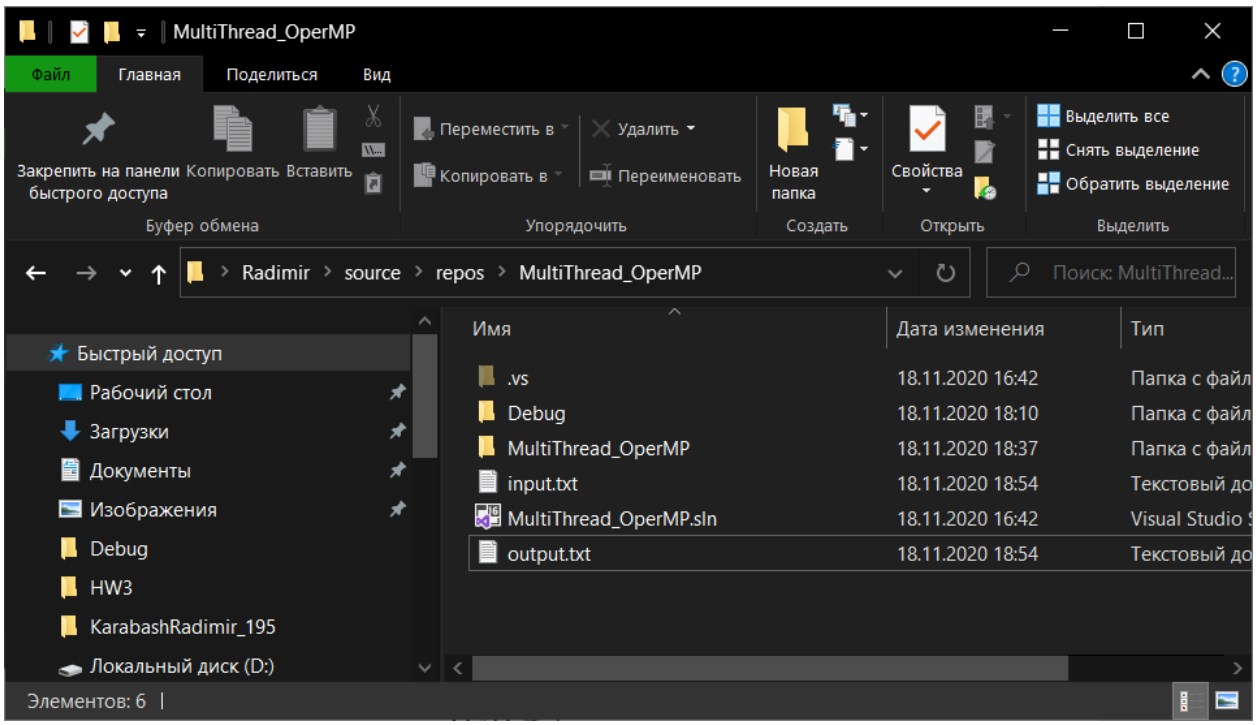


TEST 3.5

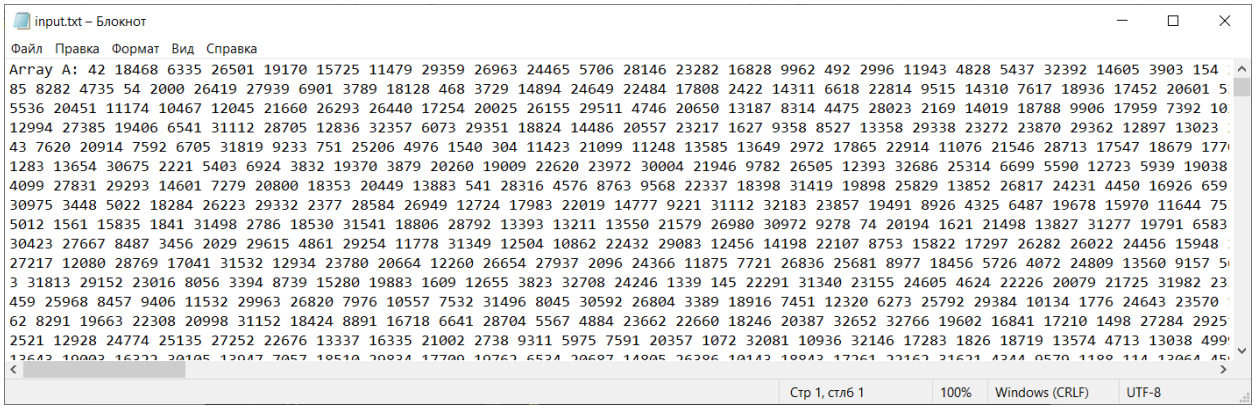


TEST 4.1

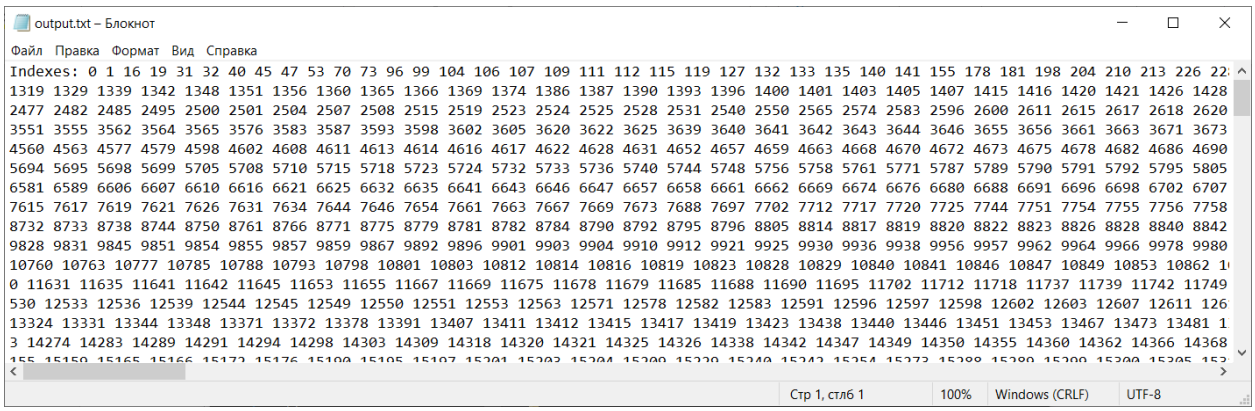
TEST 4.2



TEST 4.3



TEST 4.4



TEST 4.5