Отчет по лабораторной работе №4

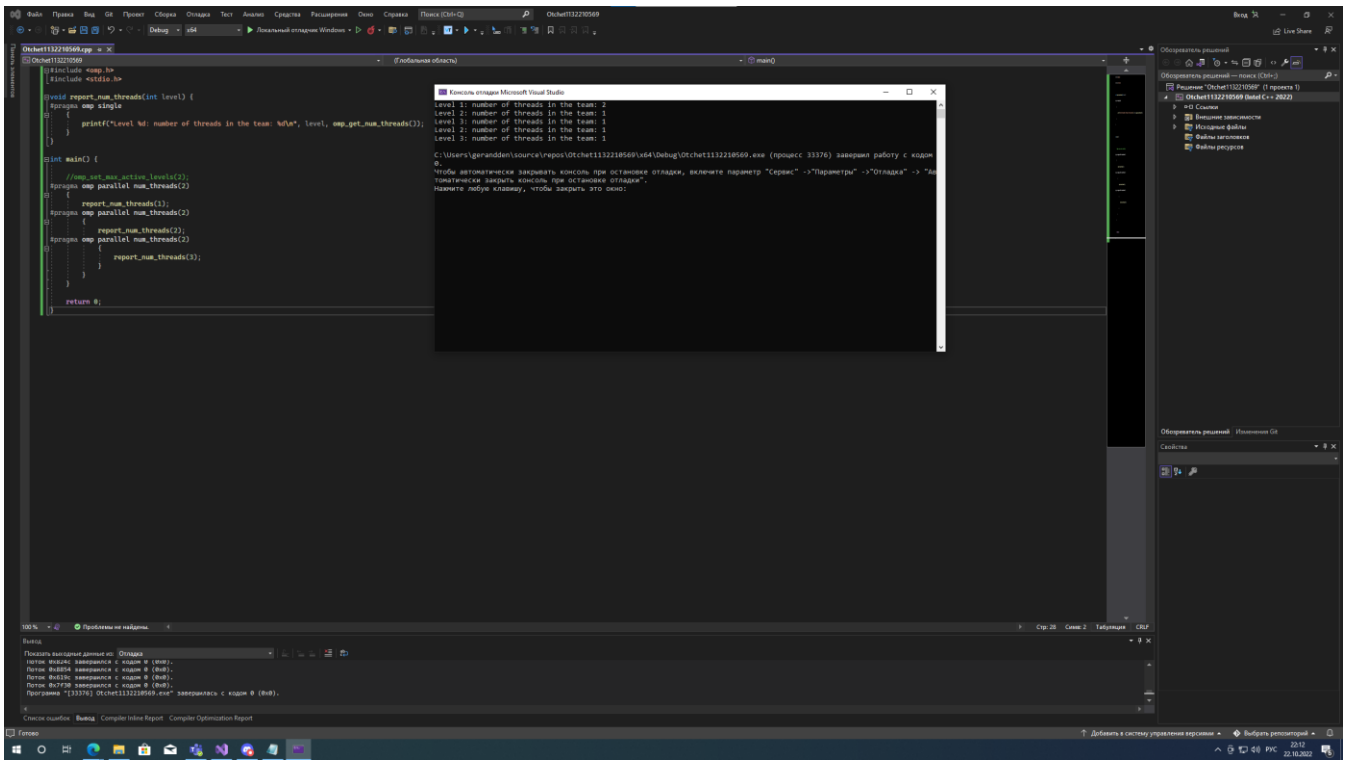
Выполнил Герасимов АД, ИУСбд-01-20, 1132210569

Example 4.1

Упражнения.

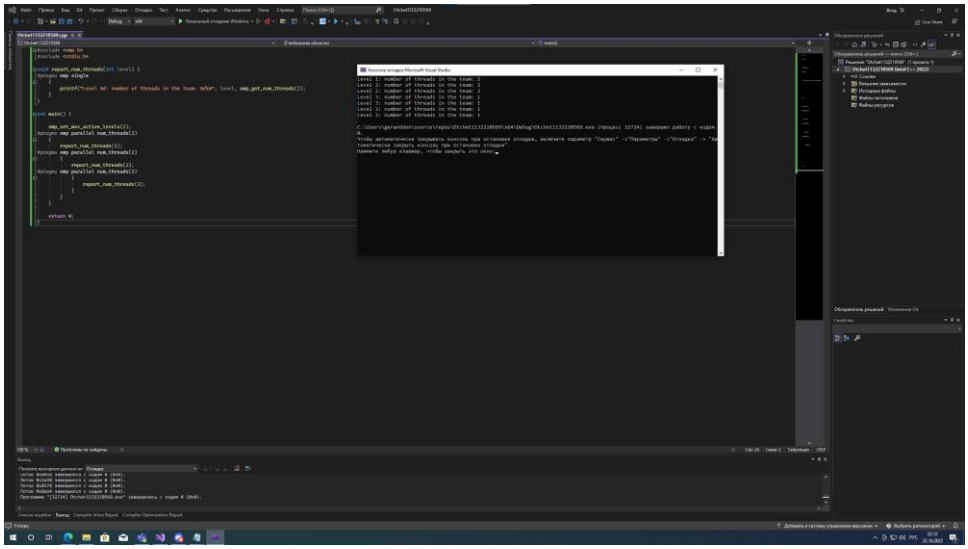
1. Запустите программу и объясните результат.
Проанализируйте и объясните назначение инструкции single в report_num_threads.

```
#include <omp.h>
#include <stdio.h>

void report_num_threads(int level) {
#pragma omp single
	{
		printf("Level %d: number of threads in the team: %d\n", level,
omp_get_num_threads());
	}
}

int main() {

#pragma omp parallel num_threads(2)
	{
		report_num_threads(1);
#pragma omp parallel num_threads(2)
		{
			report_num_threads(2);
#pragma omp parallel num_threads(2)
			{
				report_num_threads(3);
			}
		}
	}

	return 0;
}
```



2. Используйте перед первой паралельной секцией настройку переменной OMP_NESTED (переменная OMP_NESTED разрешает или запрещает вложенный параллелизм) с помощью вызова функции omp_set_nested():
   1) omp_set_nested(0);
   2) omp_set_nested(1);
   Обратите внимание на изменения в работе программы.

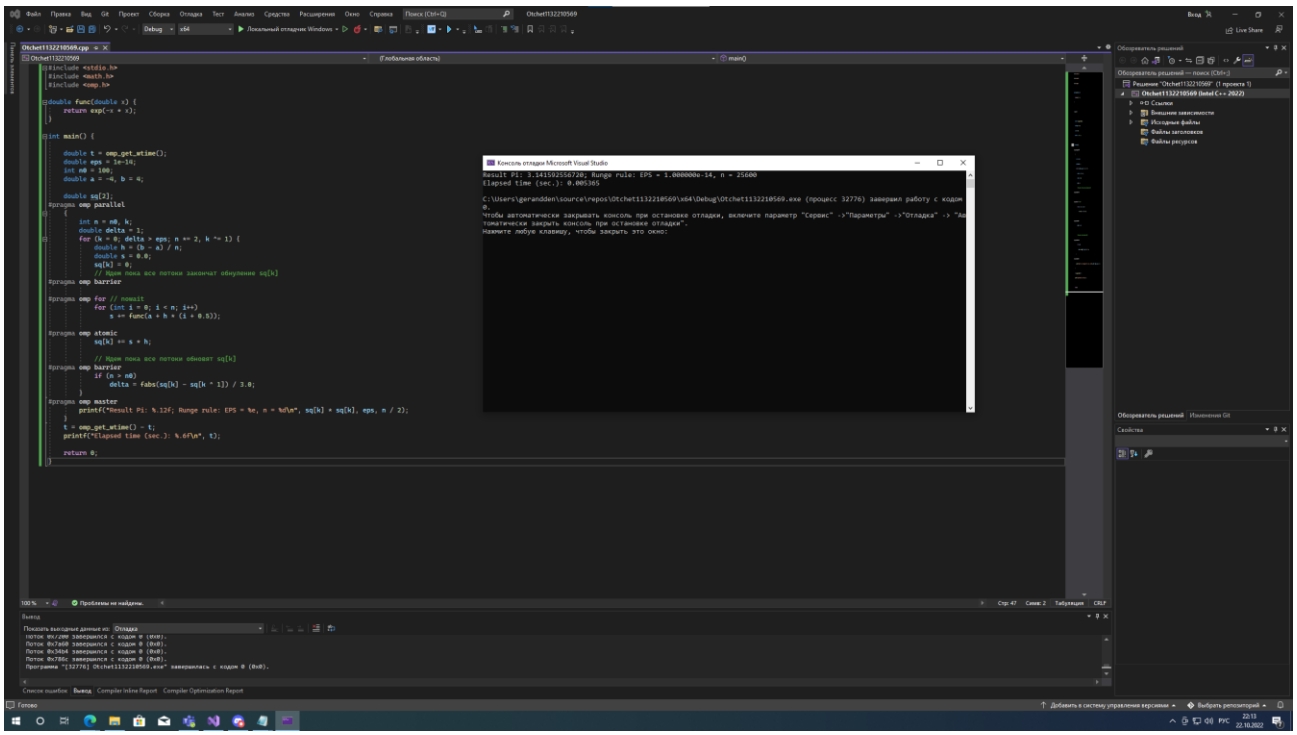
```
#include <omp.h>
#include <stdio.h>

void report_num_threads(int level) {
#pragma omp single
	{
		printf("Level %d: number of threads in the team: %d\n", level,
omp_get_num_threads());
	}
}

int main() {

	omp_set_max_active_levels(2);
#pragma omp parallel num_threads(2)
	{
		report_num_threads(1);
#pragma omp parallel num_threads(2)
		{
			report_num_threads(2);
#pragma omp parallel num_threads(2)
			{
				report_num_threads(3);
			}
		}
	}

	return 0;
}
```



Example 4.2b (integral parallel).cpp

```
#include <stdio.h>
#include <math.h>
```

```
#include <omp.h>

double func(double x) {
    return exp(-x * x);
}

int main() {

    double t = omp_get_wtime();
    double eps = 1e-14;
    int n0 = 100;
    double a = -4, b = 4;

    double sq[2];
    #pragma omp parallel
    {
        int n = n0, k;
        double delta = 1;
        for (k = 0; delta > eps; n *= 2, k ^= 1) {
            double h = (b - a) / n;
            double s = 0.0;
            sq[k] = 0;
            // Ждем пока все потоки закончат обнуление sq[k]
            #pragma omp barrier

            #pragma omp for // nowait
            for (int i = 0; i < n; i++)
                s += func(a + h * (i + 0.5));

            #pragma omp atomic
            sq[k] += s * h;

            // Ждем пока все потоки обновят sq[k]
            #pragma omp barrier
            if (n > n0)
                delta = fabs(sq[k] - sq[k ^ 1]) / 3.0;
        }
        #pragma omp master
        printf("Result Pi: %.12f; Runge rule: EPS = %e, n = %d\n", sq[k] * sq[k],
eps, n / 2);
    }
    t = omp_get_wtime() - t;
    printf("Elapsed time (sec.): %.6f\n", t);

    return 0;
}
```



/*

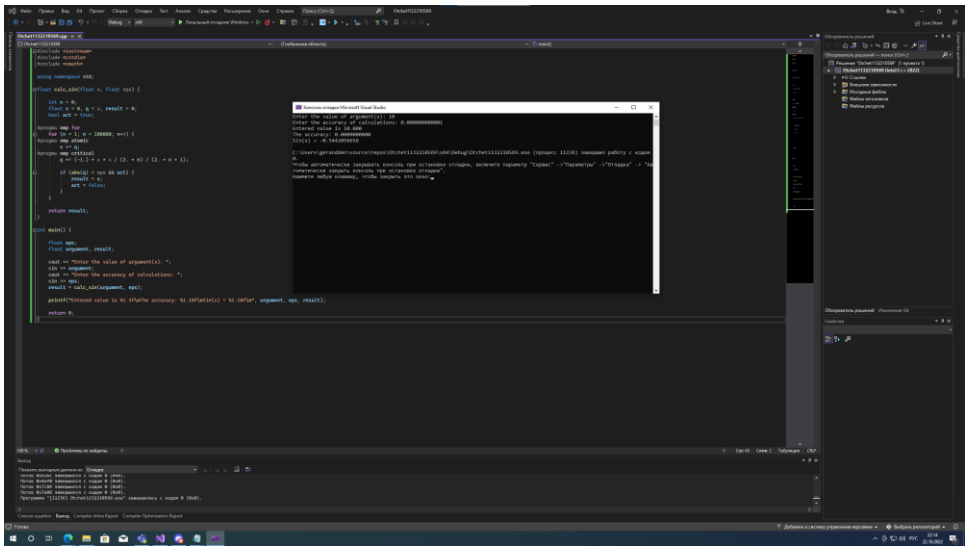
Упражнения.

1. Проанализируйте предложенный вариант распараллеливания. Есть ли альтернативные схемы распараллеливания?

2. Напишите параллельную программу вычисляющую параллельно значение одной из тригонометрических функций
    с наперёд заданной точностью с помощью формул Макларена. Точность можно оценивать как значение
    последнего вычисленного слагаемого.

```
#include <iostream>
#include <cstdio>
#include <cmath>

using namespace std;

float calc_sin(float x, float eps) {

	int n = 0;
	float s = 0, q = x, result = 0;
	bool act = true;

#pragma omp for
	for (n = 1; n < 100000; n++) {
#pragma omp atomic
		s += q;
#pragma omp critical
		q *= (-1.) * x * x / (2. * n) / (2. * n + 1);

		if (abs(q) < eps && act) {
			result = s;
			act = false;
		}
	}
```

```
        return result;
}

int main() {

        float eps;
        float argument, result;

        cout << "Enter the value of argument(x): ";
        cin >> argument;
        cout << "Enter the accuracy of calculations: ";
        cin >> eps;
        result = calc_sin(argument, eps);

        printf("Entered value is %1.3f\nThe accuracy: %1.10f\nSin(x) = %1.10f\n", argument,
eps, result);

        return 0;
}
```



```
*/
```

Домашнее задание (базовое).

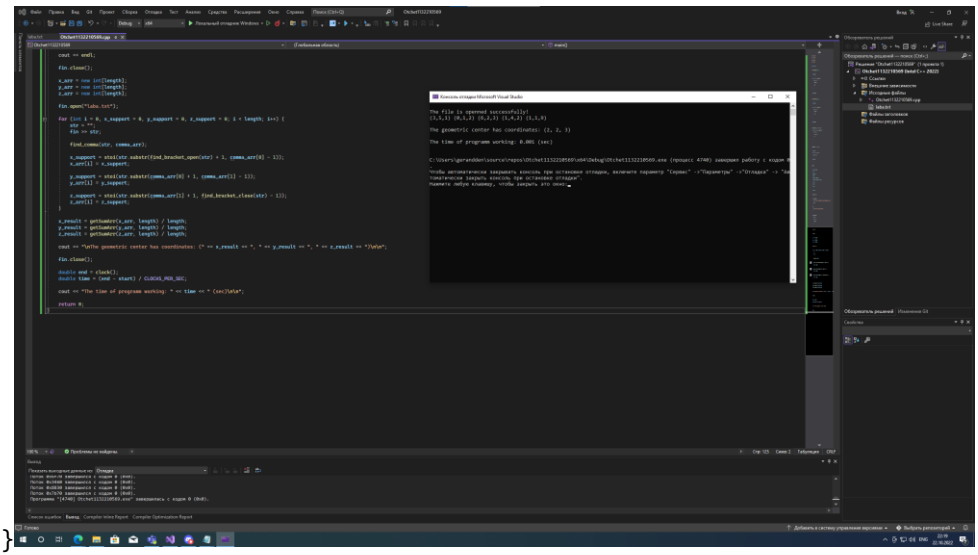
Задание 1.

Напишите программу, которая читает из файла координаты точек в 3D пространстве (x,y,z) и вычисляет геометрический центр, который есть среднее по x, y и z. Напишите две версии программы: одну с распараллеливанием цикла, другую на основе функциональной декомпозиции. Определите наиболее рациональную схему из критерия быстродействия программы. Используйте два варианта чтения: полная загрузка содержания файла и работа с файлом в режиме прямого доступа. Насколько сильно на быстродействие влияет формат чтения?

```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>

using namespace std;

int find_bracket_open(string str) {

        int index_bop = 0;
```

```
	for (int i = 0; i < str.length(); i++) {
		if (str[i] == '(') {
			index_bop = i;
			break;
		}
	}

	return index_bop;
}

int find_bracket_close(string str) {

	int index_bcl = 0;

	for (int i = 0; i < str.length(); i++) {
		if (str[i] == ')') {
			index_bcl = i;
			break;
		}
	}

	return index_bcl;
}

void find_comma(string str, int comma_arr[]) {
	for (int i = 0, k = 0; i < str.length(); i++) {
		if (str[i] == ',') {
			comma_arr[k] = i;
			k++;
		}
	}
}

int getSumArr(int* arr, int length) {

	int result = 0;
	for (int i = 0; i < length; ++i) {
		result += arr[i];
	}
	return result;
}

int main() {

	double start = clock();
	char ch;
	string points = "___";
	string str;
	int length = 1, comma_arr[2];
	int x_result = 0, y_result = 0, z_result = 0;
	int* x_arr;
	int* y_arr;
	int* z_arr;

	ifstream fin;

	fin.open("laba.txt");

	if (!fin.is_open()) {
		cout << "\nThe file is NOT openned!\nTry to launch the prorgamm one more
time...\n";
		return 0;
	}
	else {
```

```
		cout << "\nThe file is openned successfully!\n";
	}

	while (fin.get(ch)) {
		cout << ch;
		if (ch == ' ') {
			length++;
		}
	}

	cout << endl;

	fin.close();

	x_arr = new int[length];
	y_arr = new int[length];
	z_arr = new int[length];

	fin.open("laba.txt");

	for (int i = 0, x_support = 0, y_support = 0, z_support = 0; i < length; i++) {
		str = "";
		fin >> str;

		find_comma(str, comma_arr);

		x_support = stoi(str.substr(find_bracket_open(str) + 1, comma_arr[0] - 1));
		x_arr[i] = x_support;

		y_support = stoi(str.substr(comma_arr[0] + 1, comma_arr[1] - 1));
		y_arr[i] = y_support;

		z_support = stoi(str.substr(comma_arr[1] + 1, find_bracket_close(str) - 1));
		z_arr[i] = z_support;
	}

	x_result = getSumArr(x_arr, length) / length;
	y_result = getSumArr(y_arr, length) / length;
	z_result = getSumArr(z_arr, length) / length;

	cout << "\nThe geometric center has coordinates: (" << x_result << ", " << y_result
<< ", " << z_result << ")\n\n";

	fin.close();

	double end = clock();
	double time = (end - start) / CLOCKS_PER_SEC;

	cout << "The time of programm working: " << time << " (sec)\n\n";

	return 0;
```

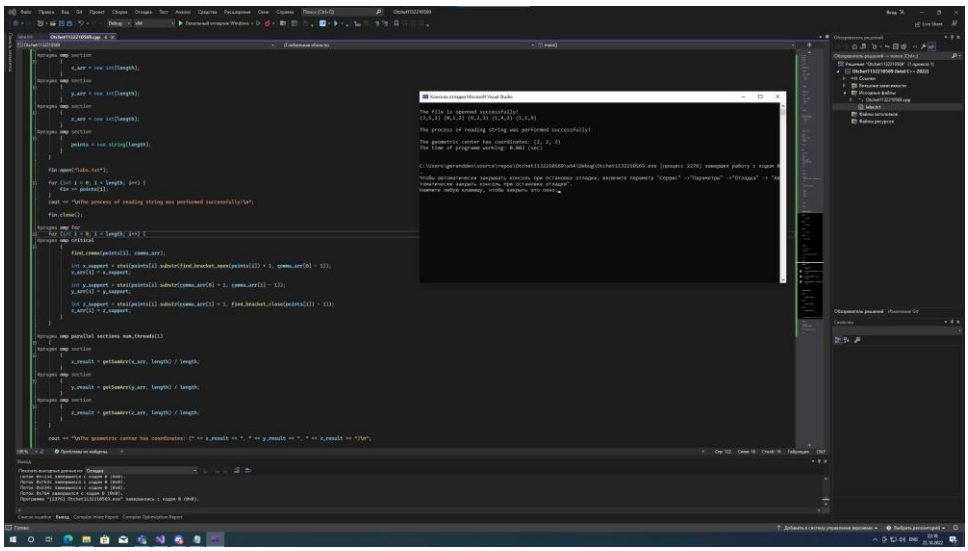


}

```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>

using namespace std;

int find_bracket_open(string str) {

	int index_bop = 0;

	for (int i = 0; i < str.length(); i++) {
		if (str[i] == '(') {
			index_bop = i;
			break;
		}
	}

	return index_bop;
}

int find_bracket_close(string str) {

	int index_bcl = 0;

	for (int i = 0; i < str.length(); i++) {
		if (str[i] == ')') {
			index_bcl = i;
			break;
		}
	}

	return index_bcl;
}

void find_comma(string str, int comma_arr[]) {
	for (int i = 0, k = 0; i < str.length(); i++) {
		if (str[i] == ',') {
			comma_arr[k] = i;
			k++;
		}
	}
}
```

```
int getSumArr(int* arr, int length) {

	int result = 0;
#pragma omp for
	for (int i = 0; i < length; ++i) {
#pragma omp atomic
		result += arr[i];
	}
	return result;
}

int main() {

	double start = clock();
	char ch;
	string* points;
	string str;
	int length = 1, comma_arr[2];
	int x_result = 0, y_result = 0, z_result = 0;
	int* x_arr;
	int* y_arr;
	int* z_arr;

	ifstream fin;

	fin.open("laba.txt");

	if (!fin.is_open()) {
		cout << "\nThe file is NOT openned!\nTry to launch the prorgamm one more
time...\n";
		return 0;
	}
	else {
		cout << "\nThe file is openned successfully!\n";
	}

	while (fin.get(ch)) {
		cout << ch;
		if (ch == ' ') {
			length++;
		}
	}

	cout << endl;

	fin.close();


#pragma omp parallel sections num_threads(3)
	{
#pragma omp section
		{
			x_arr = new int[length];
		}
#pragma omp section
		{
			y_arr = new int[length];
		}
#pragma omp section
		{
			z_arr = new int[length];
		}
#pragma omp section
		{
			points = new string[length];
```

```
			}
		}

		fin.open("laba.txt");

		for (int i = 0; i < length; i++) {
			fin >> points[i];
		}
		cout << "\nThe process of reading string was performed successfully!\n";

		fin.close();

#pragma omp for
		for (int i = 0; i < length; i++) {
#pragma omp critical
			{
				find_comma(points[i], comma_arr);

				int x_support = stoi(points[i].substr(find_bracket_open(points[i]) + 1,
comma_arr[0] - 1));
				x_arr[i] = x_support;

				int y_support = stoi(points[i].substr(comma_arr[0] + 1, comma_arr[1] -
1));
				y_arr[i] = y_support;

				int z_support = stoi(points[i].substr(comma_arr[1] + 1,
find_bracket_close(points[i]) - 1));
				z_arr[i] = z_support;
			}
		}

#pragma omp parallel sections num_threads(3)
		{
#pragma omp section
			{
				x_result = getSumArr(x_arr, length) / length;
			}
#pragma omp section
			{
				y_result = getSumArr(y_arr, length) / length;
			}
#pragma omp section
			{
				z_result = getSumArr(z_arr, length) / length;
			}
		}

		cout << "\nThe geometric center has coordinates: (" << x_result << ", " << y_result
<< ", " << z_result << ")\n";

		fin.close();

		double end = clock();
		double time = (end - start) / CLOCKS_PER_SEC;

		cout << "The time of programm working: " << time << " (sec)\n\n";

		return 0;
}
```



Задание 2.

Используя функциональную декомпозицию перепишите предыдущую задачу используя вычисления по формуле:

(Σx+Σy+Σz)/3N

где N – количество точек. Используйте глобальную сумму и критические секции.

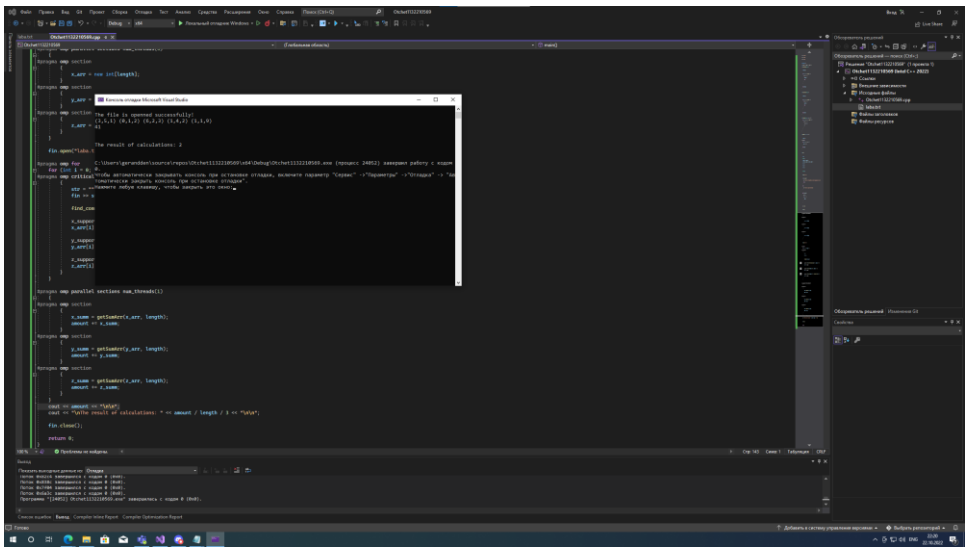
```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;
int x_support = 0, y_support = 0, z_support = 0;
int find_bracket_open(string str) {

int index_bop = 0;

for (int i = 0; i < str.length(); i++) {
if (str[i] == '(') {
index_bop = i;
break;
}
}

return index_bop;
}
```

```
int find_bracket_close(string str) {

int index_bcl = 0;

for (int i = 0; i < str.length(); i++) {
if (str[i] == ')') {
index_bcl = i;
break;
}
}

return index_bcl;
}

void find_comma(string str, int comma_arr[]) {
for (int i = 0, k = 0; i < str.length(); i++) {
if (str[i] == ',') {
comma_arr[k] = i;
k++;
}
}
}

int getSumArr(int* arr, int length) {

int result = 0;
#pragma omp for
for (int i = 0; i < length; ++i) {
result += arr[i];
}
return result;
}

int main() {
```

```
char ch;
string str;
int length = 1, comma_arr[2];
int x_summ = 0, y_summ = 0, z_summ = 0, amount = 0;
int* x_arr;
int* y_arr;
int* z_arr;

ifstream fin;

fin.open("laba.txt");

if (!fin.is_open()) {
cout << "\nThe file is NOT openned!\nTry to launch the prorgamm one more time...\n";
return 0;
}
else {
cout << "\nThe file is openned successfully!\n";
}

while (fin.get(ch)) {
cout << ch;
if (ch == ' ') {
length++;
}
}

cout << endl;

fin.close();

#pragma omp parallel sections num_threads(3)
{
#pragma omp section
{
```

```
x_arr = new int[length];
}
#pragma omp section
{
y_arr = new int[length];
}
#pragma omp section
{
z_arr = new int[length];
}
}

fin.open("laba.txt");

#pragma omp for
for (int i = 0; i < length; i++) {
#pragma omp critical
{
str = "";
fin >> str;

find_comma(str, comma_arr);

x_support = stoi(str.substr(find_bracket_open(str) + 1, comma_arr[0] - 1));
x_arr[i] = x_support;

y_support = stoi(str.substr(comma_arr[0] + 1, comma_arr[1] - 1));
y_arr[i] = y_support;

z_support = stoi(str.substr(comma_arr[1] + 1, find_bracket_close(str) - 1));
z_arr[i] = z_support;
}
}

#pragma omp parallel sections num_threads(1)
```

```
{
#pragma omp section
{
x_summ = getSumArr(x_arr, length);
amount += x_summ;
}
#pragma omp section
{
y_summ = getSumArr(y_arr, length);
amount += y_summ;
}
#pragma omp section
{
z_summ = getSumArr(z_arr, length);
amount += z_summ;
}
}
cout << amount << "\n\n";
cout << "\nThe result of calculations: " << amount / length / 3 << "\n\n";

fin.close();

return 0;
}
```



Задание 3.

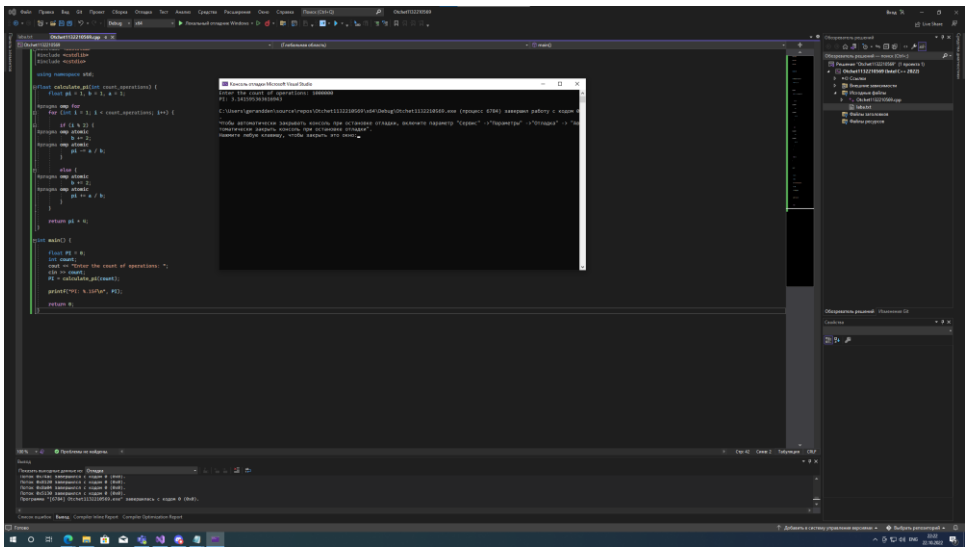
Вычислить число пи с помощью одной из формул:

· Формула Мадхавы-Лейбница (15 век)

· Формула Валлиса (17 век)

используя параллельную работу соответствующих циклов.

```
#include <iostream>
#include <cstdlib>
#include <cstdio>

using namespace std;

float calculate_pi(int count_operations) {
	float pi = 1, b = 1, a = 1;

#pragma omp for
	for (int i = 1; i < count_operations; i++) {

		if (i % 2) {
#pragma omp atomic
			b += 2;
#pragma omp atomic
			pi -= a / b;
		}

		else {
#pragma omp atomic
			b += 2;
#pragma omp atomic
			pi += a / b;
		}
	}

	return pi * 4;
}

int main() {

	float PI = 0;
	int count;
	cout << "Enter the count of operations: ";
	cin >> count;
	PI = calculate_pi(count);

	printf("PI: %.15f\n", PI);

	return 0;
}
```

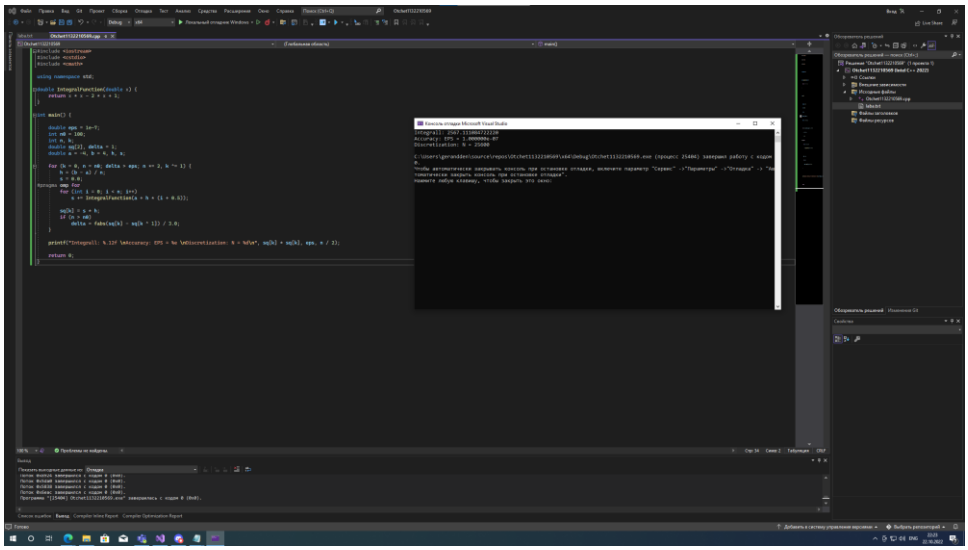


Задание 4.

Реализуйте один из методов вычисления определённого интеграла из главы 3. Вычисление определённых интегралов книги Программирование и информатика Антонюк В.А., Иванов А.П., 2015 в параллельном варианте. И протестируйте его. Используйте правило Рунге (https://ru.wikipedia.org/wiki/Правило_Рунге) для получения значения определённого интеграла с заданной точностью.

```
#include <iostream>
#include <cstdio>
#include <cmath>

using namespace std;

double IntegralFunction(double x) {
	return x * x - 2 * x + 1;
}

int main() {

	double eps = 1e-7;
	int n0 = 100;
	int n, k;
	double sq[2], delta = 1;
	double a = -4, b = 4, h, s;

	for (k = 0, n = n0; delta > eps; n *= 2, k ^= 1) {
		h = (b - a) / n;
		s = 0.0;
#pragma omp for
		for (int i = 0; i < n; i++)
			s += IntegralFunction(a + h * (i + 0.5));

		sq[k] = s * h;
		if (n > n0)
			delta = fabs(sq[k] - sq[k ^ 1]) / 3.0;
	}

	printf("Integrall: %.12f \nAccuracy: EPS = %e \nDiscretization: N = %d\n", sq[k] *
sq[k], eps, n / 2);
```

```
        return 0;
}
```

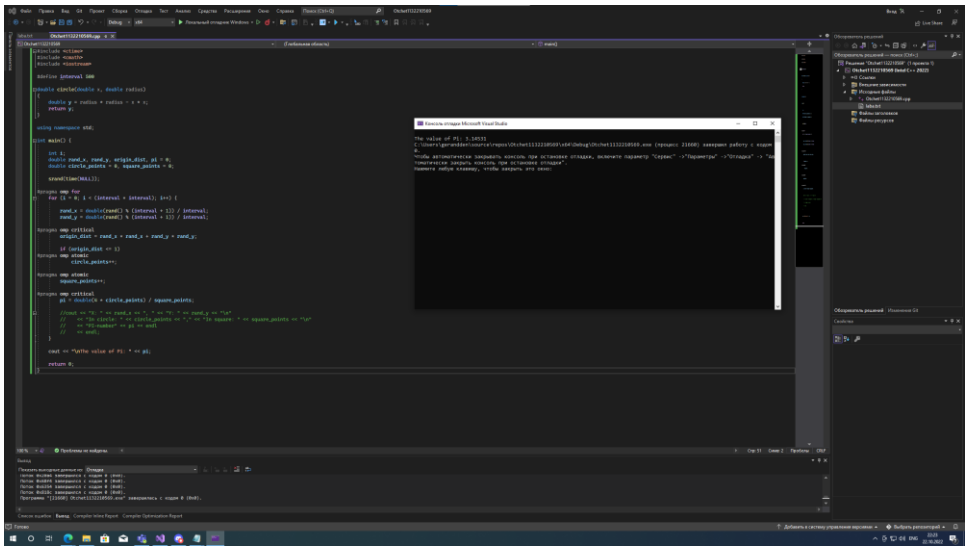


Задание 5.

Реализуйте метод определения числа π с помощью метода Монте-Карло. (см. Программирование и информатика Антонюк В.А., Иванов А.П., 2015) с помощью параллельной программы. И протестируйте его.

```
#include <ctime>
#include <cmath>
#include <iostream>

#define interval 500

double circle(double x, double radius)
{
    double y = radius * radius - x * x;
    return y;
}

using namespace std;

int main() {

    int i;
    double rand_x, rand_y, origin_dist, pi = 0;
    double circle_points = 0, square_points = 0;

    srand(time(NULL));

#pragma omp for
    for (i = 0; i < (interval * interval); i++) {

        rand_x = double(rand() % (interval + 1)) / interval;
        rand_y = double(rand() % (interval + 1)) / interval;

#pragma omp critical
        origin_dist = rand_x * rand_x + rand_y * rand_y;

        if (origin_dist <= 1)
```

```
#pragma omp atomic
            circle_points++;

#pragma omp atomic
        square_points++;

#pragma omp critical
        pi = double(4 * circle_points) / square_points;

        //cout << "X: " << rand_x << ", " << "Y: " << rand_y << "\n"
        //    << "In circle: " << circle_points << "," << "In square: " << square_points <<
"\n"
        //    << "PI-number" << pi << endl
        //    << endl;
    }

    cout << "\nThe value of Pi: " << pi;

    return 0;
}
```



Задание 6.

Изучите код приведённой программы, запустите и сравните результаты. Проведите распараллеливание циклов с и без контроля последовательности вычислений. Проанализируйте результат.

void summation_sequence() {

int j;

float summ = 0;

for (j = 1; j <= 1000; j++)

summ += 1.0 / j;

printf("Сумма равна %12.9f\n", summ);

summ = 0;

for (j = 1000; j >= 1; j--)

```
summ += 1.0 / j;

printf("Сумма равна %12.9f\n", summ);

}
```

```
#include <stdio.h>

void summation_sequence() {

	int j;
	float summ = 0;

#pragma omp for
	for (j = 1; j <= 1000; j++) {
#pragma omp critical
		{
			summ += 1.0 / j;
		}
	}

	printf("The amount:%12.9f\n", summ);

	summ = 0;

#pragma omp for
	for (j = 1000; j >= 1; j--) {
#pragma omp critical
		{
			summ += 1.0 / j;
		}
	}

	printf("The amount:%12.9f\n", summ);

}

int main() {

	summation_sequence();

	return 0;
}
```