Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

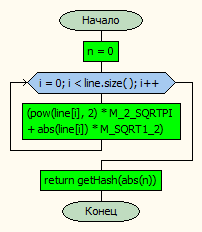
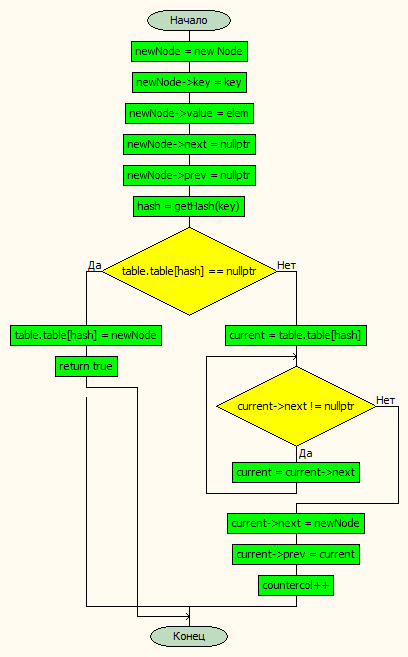
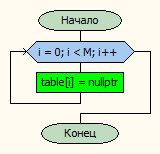
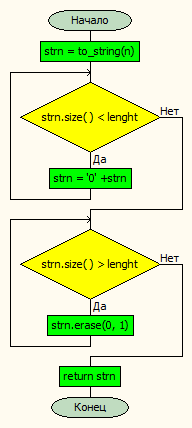
Выполнил:   
студент группы РИС-23-1б   
Молодых Никита Андреевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

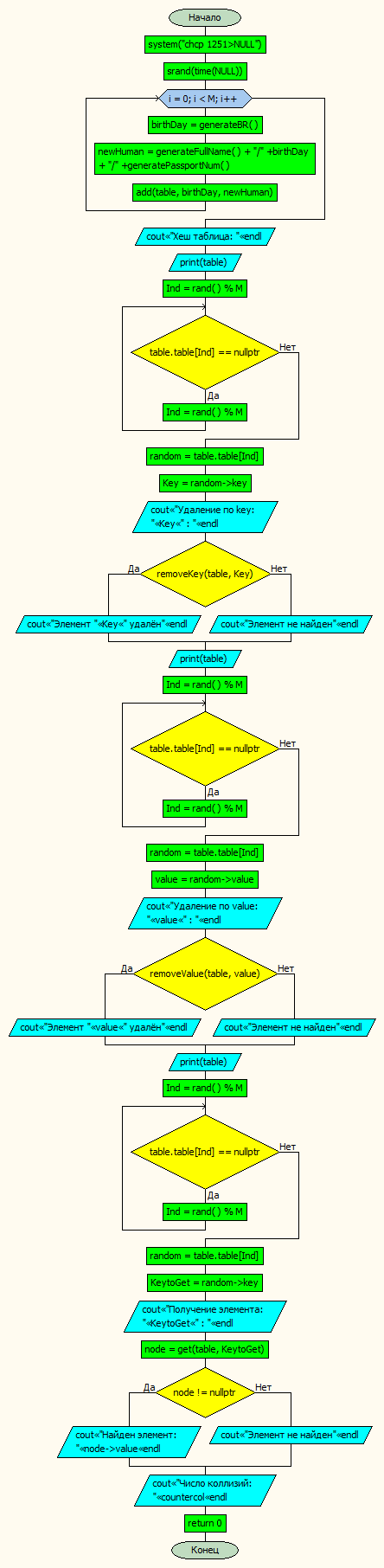
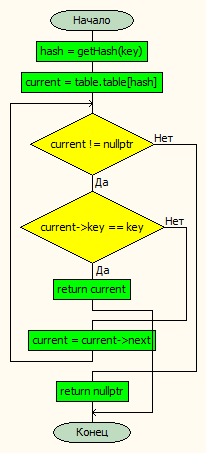
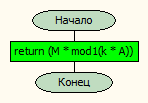
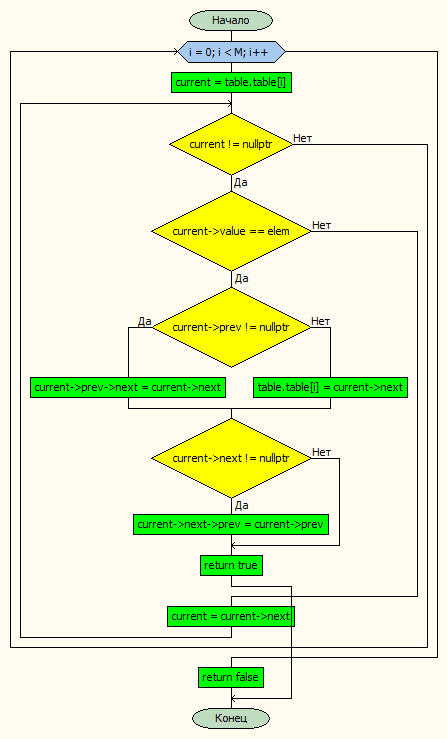
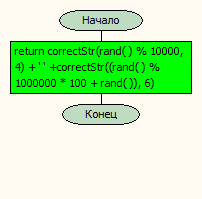
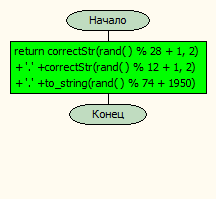
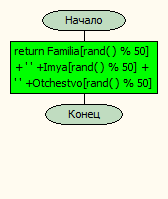
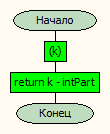
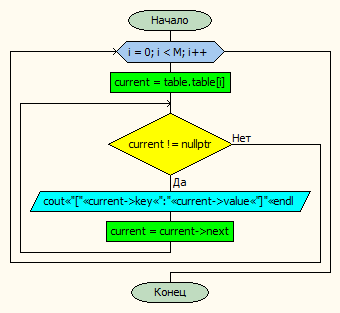
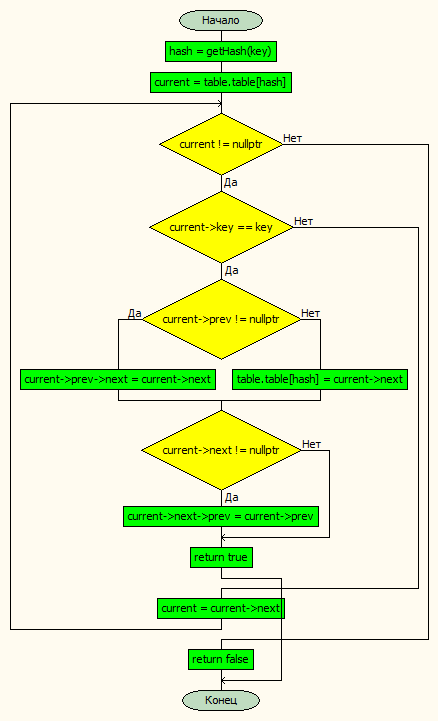
Пермь, 2024 г.

**«Сортировки»**

**«**Многофазная**»**



**»**



**Блок-Схема:**

**Код на языке C++:**

#include <iostream>

#include <fstream>

using namespace std;

// Структура, представляющая отсортированную серию целых чисел

struct Series

{

int\* elements;

int lenght;

void PrintSeriers()

{

if (lenght <= 0)

return;

cout << elements[0];

for (int i = 1; i < lenght; i++)

cout << " " << elements[i];

cout << endl;

}

};

struct Bucket

{

Series\* seriesArr;

int lenght;

};

void HoareQuickSorting(int\* Arr, int startIndex, int endIndex)

{

if (startIndex <= endIndex)

{

int middleElement = Arr[(startIndex + endIndex) / 2];

int LeftIndex = startIndex;

int RightIndex = endIndex;

while (LeftIndex <= RightIndex)

{

while (Arr[LeftIndex] < middleElement)

LeftIndex++;

while (Arr[RightIndex] > middleElement)

RightIndex--;

if (LeftIndex <= RightIndex)

{

int tmp = Arr[LeftIndex];

Arr[LeftIndex] = Arr[RightIndex];

Arr[RightIndex] = tmp;

LeftIndex++;

RightIndex--;

}

}

HoareQuickSorting(Arr, startIndex, RightIndex);

HoareQuickSorting(Arr, LeftIndex, endIndex);

}

}

void HoareQuickSorting(int\* Arr, int Lenght)

{

if (Lenght > 1)

HoareQuickSorting(Arr, 0, Lenght - 1);

}

Series MS(Series& SeriesOne, Series& SeriesTwo)

{

Series NewSeries;

NewSeries.elements = new int[SeriesOne.lenght + SeriesTwo.lenght];

for (int i = 0; i < SeriesOne.lenght; i++)

NewSeries.elements[i] = SeriesOne.elements[i];

int counter = SeriesOne.lenght;

for (int i = 0; i < SeriesTwo.lenght; i++)

{

NewSeries.elements[counter] = SeriesTwo.elements[i];

counter++;

}

NewSeries.lenght = SeriesOne.lenght + SeriesTwo.lenght;

HoareQuickSorting(NewSeries.elements, NewSeries.lenght);

return NewSeries;

}

// Функция для получения пары чисел Фибоначчи

pair<int, int> GetFibonnachiNumbers(int numberToCompareTo, pair<int, int> PreviousTwoNumbers)

{

int currentNumber = PreviousTwoNumbers.first + PreviousTwoNumbers.second;

pair <int, int> PairToReturn;

if (currentNumber < numberToCompareTo)

PairToReturn = GetFibonnachiNumbers(numberToCompareTo, { PreviousTwoNumbers.second, currentNumber });

else

PairToReturn = PreviousTwoNumbers;

return PairToReturn;

}

// Перегрузка функции для получения чисел Фибоначчи без предыдущих значений

pair<int, int> GetFibonnachiNumbers(int numberToCompareTo)

{

return GetFibonnachiNumbers(numberToCompareTo, { 0, 1 });

}

// Основная функция сортировки слиянием

void SortUsingManyPhaseMerging(ifstream\* file, int Lenght)

{

cout << "Начало сортировки!" << endl;

// Создание массива серий

Series\* SeriesArr = new Series[Lenght]; //first - серия, second - ее длина

int AmountOfSeries = 1;

int indexOfCurSeries = 0;

SeriesArr[0].elements = new int[Lenght];

SeriesArr[0].lenght = 1;

// Чтение первого элемента из файла

\*file >> SeriesArr[0].elements[0];

// Чтение остальных элементов из файла и разделение их на серии

for (int i = 1; i < Lenght; i++)

{

int curElement;

\*file >> curElement;

if (curElement >= SeriesArr[indexOfCurSeries].elements[SeriesArr[indexOfCurSeries].lenght - 1])

{

// Добавление элемента в текущую серию, если он больше или равен последнему элементу серии

SeriesArr[indexOfCurSeries].elements[SeriesArr[indexOfCurSeries].lenght] = curElement;

SeriesArr[indexOfCurSeries].lenght++;

}

else

{

// Создание новой серии, если текущий элемент меньше последнего элемента текущей серии

indexOfCurSeries++;

AmountOfSeries++;

SeriesArr[indexOfCurSeries].elements = new int[Lenght];

SeriesArr[indexOfCurSeries].lenght = 1;

SeriesArr[indexOfCurSeries].elements[0] = curElement;

}

}

// Вычисление оптимального количества серий для слияния на основе чисел Фибоначчи

pair<int, int> PerfectFibonachiNumbers = GetFibonnachiNumbers(Lenght);

// Инициализация трех бакетов

Bucket\* Buckets = new Bucket[3];

// Установка длин бакетов

Buckets[0].lenght = PerfectFibonachiNumbers.second;

Buckets[1].lenght = PerfectFibonachiNumbers.first;

Buckets[2].lenght = 0;

// Выделение памяти для серий в каждом бакете

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < Lenght; j++)

Buckets[i].seriesArr = new Series[Lenght];

}

// Выделение памяти для элементов серий в каждом бакете

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < Lenght; j++)

{

Buckets[i].seriesArr[j].elements = new int[Lenght];

Buckets[i].seriesArr[j].lenght = 0;

}

}

// Распределение серий по бакетам

indexOfCurSeries = 0;

for (int i = 0; i < 2 && indexOfCurSeries < AmountOfSeries; i++)

{

for (int j = 0; j < Buckets[i].lenght && indexOfCurSeries < AmountOfSeries; j++)

{

Buckets[i].seriesArr[j] = SeriesArr[indexOfCurSeries];

indexOfCurSeries++;

}

}

// Установка переменных для бакетов

AmountOfSeries = PerfectFibonachiNumbers.first + PerfectFibonachiNumbers.second;

int IndexOfTheBiggerBucket = 0;

int IndexOfTheSmallerBucket = 1;

int IndexOfTheHelpingBucket = 2;

// Многопроходное слияние серий

while (AmountOfSeries > 1)

{

// Вычисление новой длины

int NewLenght = Buckets[IndexOfTheBiggerBucket].lenght - Buckets[IndexOfTheSmallerBucket].lenght;

int curIndexOfTheHelpingBucket = 0;

int lastIndexOfSmallerBucket = Buckets[IndexOfTheSmallerBucket].lenght - 1;

// Слияние серий и помещение результата в вспомогательный бакет

for (int i = Buckets[IndexOfTheBiggerBucket].lenght - 1; i > NewLenght - 1; i--)

{

Series NewSeries = MS(Buckets[IndexOfTheBiggerBucket].seriesArr[i], Buckets[IndexOfTheSmallerBucket].seriesArr[lastIndexOfSmallerBucket]);

Buckets[IndexOfTheHelpingBucket].seriesArr[curIndexOfTheHelpingBucket] = NewSeries;

Buckets[IndexOfTheHelpingBucket].lenght++;

curIndexOfTheHelpingBucket++;

lastIndexOfSmallerBucket--;

AmountOfSeries--;

}

// Обновление длины бакетов

Buckets[IndexOfTheBiggerBucket].lenght -= Buckets[IndexOfTheSmallerBucket].lenght;

// Очистка памяти от старых серий в маленьком бакете

for (int i = 0; i < Buckets[IndexOfTheSmallerBucket].lenght; i++)

delete[] Buckets[IndexOfTheSmallerBucket].seriesArr[i].elements;

// Освобождение памяти от массива серий в маленьком бакете

delete[] Buckets[IndexOfTheSmallerBucket].seriesArr;

// Выделение памяти для нового массива серий в маленьком бакете

Buckets[IndexOfTheSmallerBucket].seriesArr = new Series[Lenght];

// Выделение памяти для элементов серий в новом массиве серий в маленьком бакете

for (int i = 0; i < Lenght; i++)

{

Buckets[IndexOfTheSmallerBucket].seriesArr[i].elements = new int[Lenght];

Buckets[IndexOfTheSmallerBucket].seriesArr[i].lenght = 0;

}

// Обновление длины маленького бакета

Buckets[IndexOfTheSmallerBucket].lenght = 0;

// Обмен указателями на бакеты

int tmp = IndexOfTheBiggerBucket;

IndexOfTheBiggerBucket = IndexOfTheHelpingBucket;

IndexOfTheHelpingBucket = IndexOfTheSmallerBucket;

IndexOfTheSmallerBucket = tmp;

}

// Получение отсортированной последовательности

Series sortedContent = Buckets[IndexOfTheBiggerBucket].seriesArr[0];

// Запись отсортированных данных в файл

ofstream Output("F2.txt");

Output << sortedContent.elements[0];

cout << sortedContent.elements[0];

for (int i = 1; i < sortedContent.lenght; i++)

{

cout << " " << sortedContent.elements[i];

Output << " " << sortedContent.elements[i];

}

Output.close();

// Освобождение памяти

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < Lenght; j++)

delete[] Buckets[i].seriesArr[j].elements;

delete[] Buckets[i].seriesArr;

}

delete[] Buckets;

delete[] SeriesArr;

}

int main()

{

system("chcp 1251>NULL");

srand(time(0)); // Инициализация генератора случайных чисел

int Lenght;

cout << "Режим генерации случайных элементов" << endl;

do

{

cout << "Введите количество элементов массива" << endl;

cin >> Lenght;

} while (Lenght <= 1);

ofstream Output("F1.txt"); // Открытие текстового файла для записи

for (int i = 0; i < Lenght; i++)

{

int element = rand() % 100;

cout << element << " ";

Output << element << " "; // Запись элемента в текстовый файл с пробелом

}

cout << endl;

Output.close(); // Закрытие файла

ifstream Input("F1.txt"); // Открытие текстового файла для чтения

SortUsingManyPhaseMerging(&Input, Lenght); // Вызов функции сортировки

Input.close(); // Закрытие файла

cout << endl << "Результат сортировки:" << endl << endl;

ifstream SortedInput("F2.txt"); // Открытие текстового файла с отсортированными данными

int num;

while (SortedInput >> num) // Считывание и вывод отсортированных элементов из файла

{

cout << num << " ";

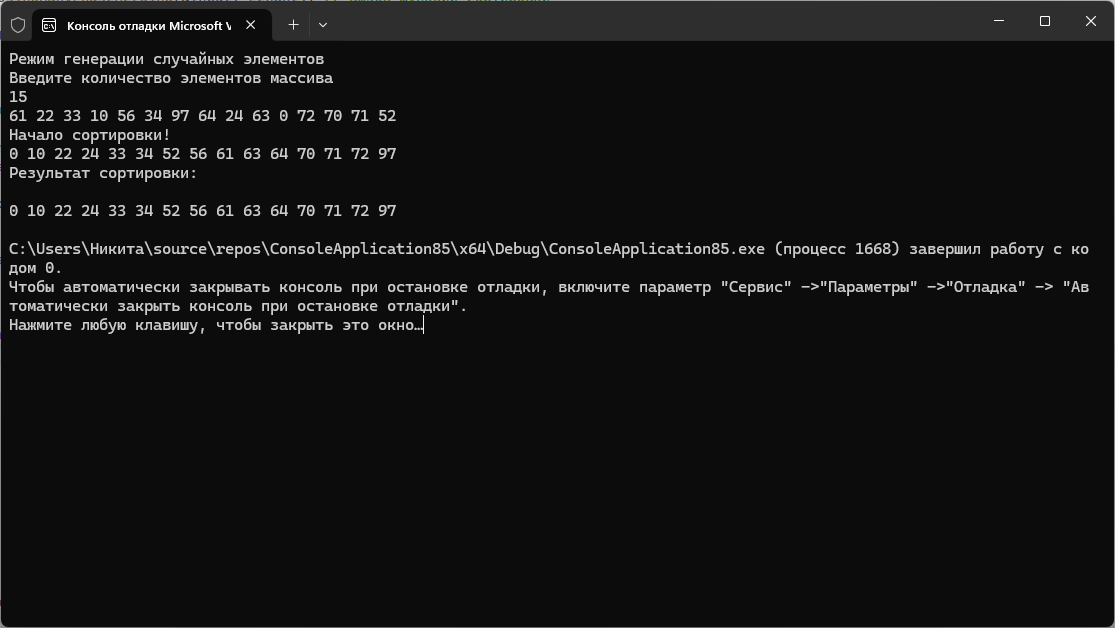
}

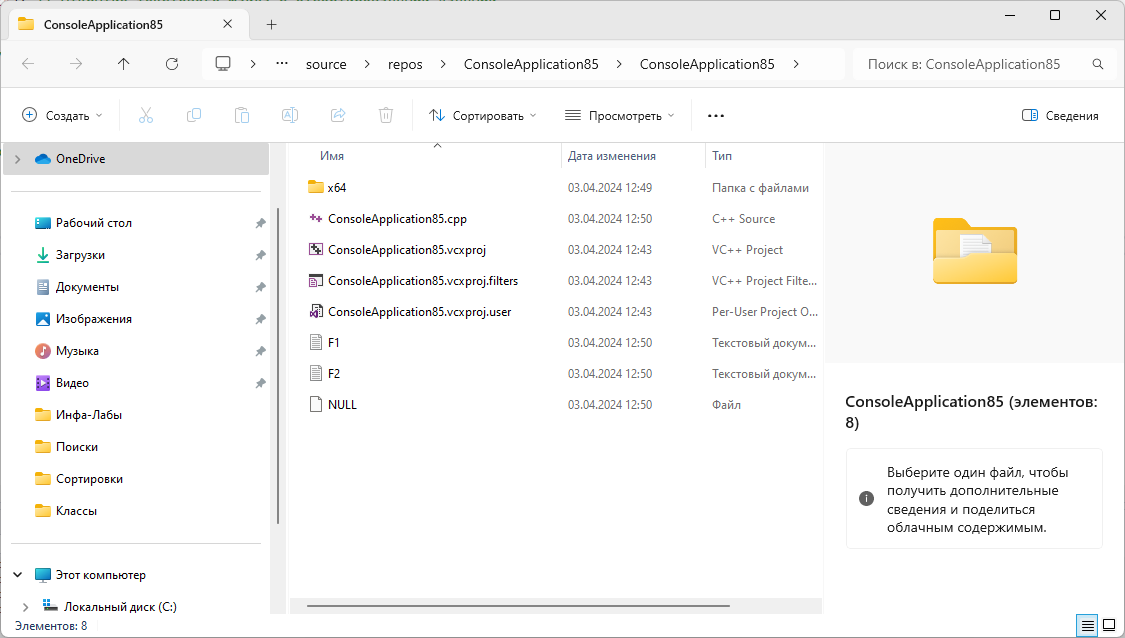
SortedInput.close(); // Закрытие файла

cout << endl;

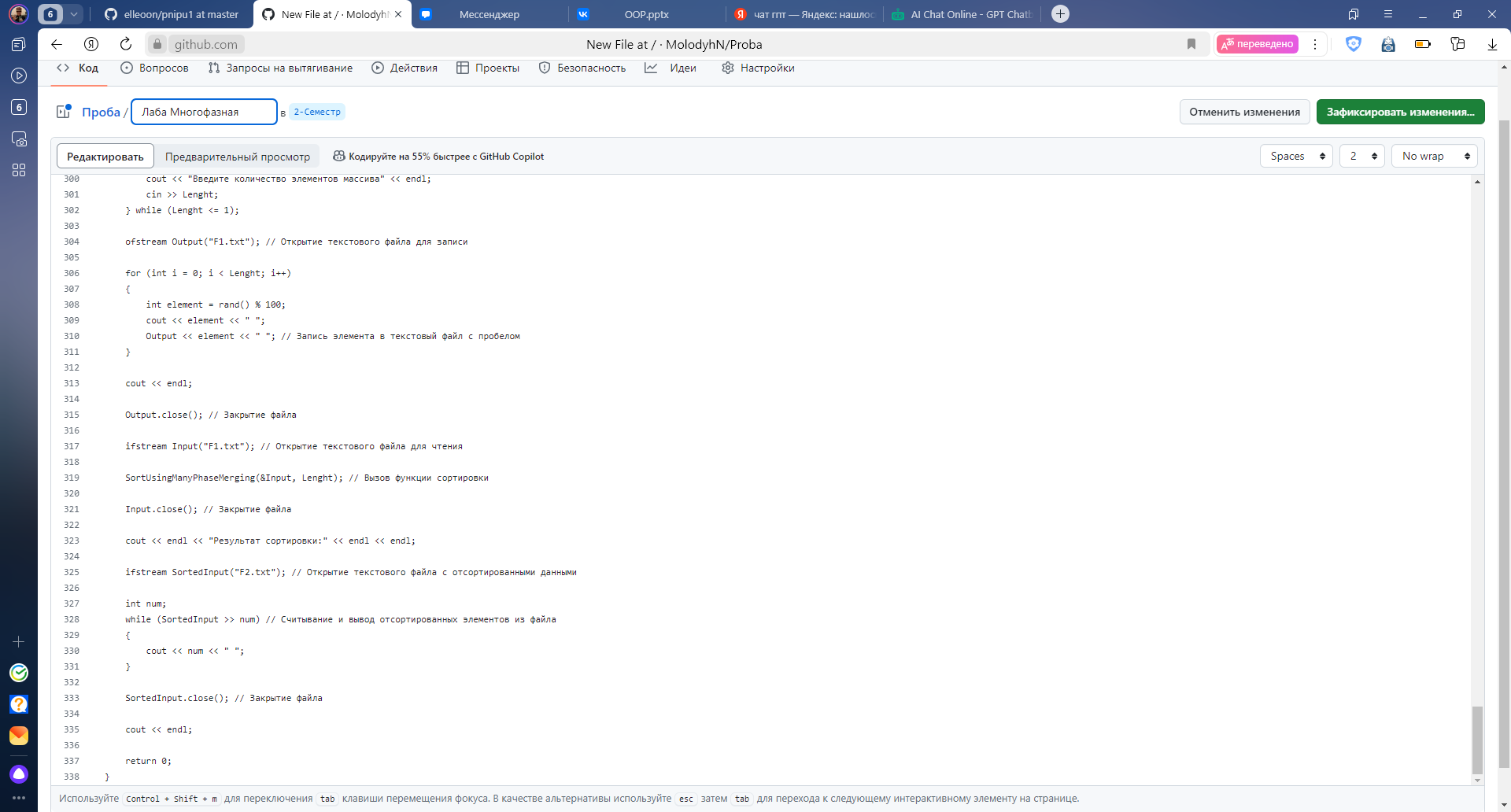
return 0;

**Работа программы:**

****

****

**Скрины из гита:**

****