

**LAB7.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <random>

#include <string>

#include <fstream>

extern int file\_max;

using namespace std;

string read\_fs(fstream& file);

bool check\_eof(iostream& strm, bool specified\_vals=false);

struct Value {

int key;

char content[196];

Value() {}

Value(int key);

void write(fstream& output);

Value static read(fstream& input);

};

class FileSort {

private:

fstream\* groupSrc;

fstream\* groupDst;

int file\_count;

bool modeA = true;

void sort\_iteration();

public:

FileSort(int file\_count);

~FileSort();

void open\_files(int file\_count, bool create = false);

void clear\_files(int file\_count);

//Возвращает отсортировано ли в A01.txt

bool sort\_A01(int& iteration\_counter);

};

/\* Пример файловой структуры:

A B

-----------------

A01.txt | B01.txt - исходный файл A01.txt и в одном из них будет результат

-----------------

A02.txt B02.txt

A03.txt B03.txt

A04.txt B04.txt

A05.txt B05.txt

\*/

class Table {

private:

int mcol;//Максимальное значение столбца

int min\_val;

int max\_val;

int drawline\_step = -1;

public:

void init\_draw();

bool draw\_line(string& line);

int specific\_column\_size = 0;

string legendr1 = "";

string legendr2 = "";

int display\_h = 40;

int length = 9;

int column\_width = 4;

int data[20] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };//2-10 путей

operator string();

};

**AiSD\_LAB7.cpp**

#include "LAB7.h"

#include <Windows.h>

#include <sstream>

#include <chrono>

typedef chrono::steady\_clock my\_clock;

my\_clock::time\_point start;

my\_clock::time\_point finish;

const int value\_count = 10000;

int file\_count = 3;

int file\_max = 10;

string specified\_values = R"V0G0N(

30

4

14

59

86

34

65

75

60

71

37

5

84

15

84

77

43

7

11

13

38

69

42

65

34

33

33

32

24

22

)V0G0N";

bool values\_are\_specified() {

for (int i = 0; i < specified\_values.size(); i++)

if (specified\_values[i] != '\n') return true;

return false;

}

void randomize() {

fstream file = fstream("A01.txt", ios::binary | ios::in | ios::out | ios::trunc);

if (values\_are\_specified()) {

stringstream values(specified\_values, ios::binary | ios::in);

while(!check\_eof(values, true)) {

Value value;

values >> value.key;

value.write(file);

}

}

else {

for (int i = 0; i < value\_count; i++) {

Value value;

value.key = rand();

value.write(file);

}

}

file.close();

}

bool metrics = false;

int main() {

//Размер консоли на максимум

HWND console = GetConsoleWindow();

RECT r;

GetWindowRect(console, &r);

ShowWindow(console, SW\_MAXIMIZE);

CONSOLE\_FONT\_INFOEX cfi = { sizeof(CONSOLE\_FONT\_INFOEX) };

HANDLE hcsb = CreateFileA("CONOUT$", GENERIC\_WRITE | GENERIC\_READ, FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

GetCurrentConsoleFontEx(hcsb, FALSE, &cfi);

//cfi.dwFontSize.X \*= 0.95;

//cfi.dwFontSize.Y \*= 0.95;

SetCurrentConsoleFontEx(hcsb, FALSE, &cfi);

CloseHandle(hcsb);

Table table\_iter;

Table table\_time;

table\_iter.legendr1 = "ITERATION COUNT";

table\_time.legendr1 = "ELAPSED TIME (mcs)";

if (values\_are\_specified()) {

cout << "SPECIFIED VALUES: \n{";

stringstream values(specified\_values, ios::binary | ios::in);

while (!check\_eof(values, true)) {

int val;

values >> val;

cout << val << " ";

}

cout << "}\n\n";

}

//Коснуться файла, чтобы он просто был пустым.

fstream("B01.txt", ios::binary | ios::in | ios::out | ios::trunc).close();

bool fromA;//Считать результат с A01.txt, иначе B01.txt

int iteration\_counter;//Счётчик итераций для измерений

if (metrics) {

for (int i = 2; i <= file\_max; i++) {

file\_count = i;

FileSort fs(file\_count);

randomize();

start = my\_clock::now();

fromA = fs.sort\_A01(iteration\_counter);

finish = my\_clock::now();

long long timer = chrono::duration\_cast<chrono::microseconds>(finish - start).count();

table\_iter.data[i - 2] = iteration\_counter;

table\_time.data[i - 2] = timer;

}

string s1, s2;

table\_iter.init\_draw();

table\_time.init\_draw();

while (table\_iter.draw\_line(s1) + table\_time.draw\_line(s2))

cout << s1 << s2 << endl;

}

else {

FileSort fs(file\_count);

randomize();

fromA = fs.sort\_A01(iteration\_counter);

}

if (values\_are\_specified()) {

cout << "SORTED TO " << (char)('A' + !fromA) << "01.txt" << endl;

cout << "Continue to start reading" << endl;

system("pause");

fstream sorted;

if (fromA)

sorted = fstream("A01.txt", ios::binary | ios::in);

else

sorted = fstream("B01.txt", ios::binary | ios::in);

cout << endl;

unsigned char c[4];

int v;

while ((v = sorted.get()) != -1) {

c[0] = v;

c[1] = sorted.get();

c[2] = sorted.get();

c[3] = sorted.get();

cout << \*(int\*)c << ' ';

for (int i = 0; i < sizeof(Value::content); i++) sorted.get();

}

}

/\*for (int i = 0; i <= file\_max - 2; i++) {

cout << table\_iter.data[i] << "\n ";

}

cout << endl;

for (int i = 0; i <= file\_max-2; i++) {

cout << table\_time.data[i] << "\n ";

}\*/

return 0;

}

**definitions.cpp**

#include "LAB7.h"

//ДЛЯ ОТЛАДКИ

string read\_fs(fstream& file) {

int len = file.tellg();

file.clear();

file.seekg(0, ios\_base::beg);

string a = "";

char c[4];

while (!check\_eof(file)) {

if (file.tellg() == len) a += '|';

file.get(c[0]);

file.get(c[1]);

file.get(c[2]);

file.get(c[3]);

a += to\_string(\*(int\*)c) + " ";

for (int i = 0; i < sizeof(Value::content); i++) file.get();

}

file.clear();

file.seekg(len, ios\_base::beg);

return a;

}

bool check\_eof(iostream& strm, bool specified\_vals) {

int len = strm.tellg();

int c;

if (specified\_vals)

while ((c = strm.get()) == '\n');

else

c = strm.get();

bool result = c == -1;

strm.clear();

strm.seekg(len, ios\_base::beg);

return result;

}

Value::Value(int key) {

this->key = key;

for (int i = 0; i < sizeof(content); i++)

this->content[i] = '\0';

}

void Value::write(fstream& output) {

int its = 0;

output.write((char\*)this, sizeof(Value));

}

Value Value::read(fstream& input) {

char inp[sizeof(Value)];

input.read(inp, sizeof(Value));

return \*(Value\*)inp;

}

void FileSort::open\_files(int file\_count, bool create) {

char filename[8] = "A01.txt";

filename[0] += !modeA;

for (int i = 0; i < file\_count; i++) {

if (create && !((filename[0] == 'A' + !modeA) && filename[1] == '0' && filename[2] == '1'))

groupSrc[i] = fstream(filename, ios::binary | ios::in | ios::out | ios::trunc);

else

groupSrc[i] = fstream(filename, ios::binary | ios::in | ios::out);

filename[0] += modeA\*2-1;

groupDst[i] = fstream(filename, ios::binary | ios::in | ios::out | ios::trunc);

filename[0] -= modeA\*2-1;

if (++filename[2] > '9') {

filename[1]++;

filename[2] = '0';

}

}

}

void FileSort::clear\_files(int file\_count){

char filename[8] = "A00.txt";

for (int i = 0; i < file\_count; i++) {

if (groupSrc[i].is\_open()) groupSrc[i].close();

if (groupDst[i].is\_open()) groupDst[i].close();

if (++filename[2] > '9') {

filename[1]++;

filename[2] = '0';

}

if (filename[1] != '0' || filename[2] != '1') {

remove(filename);

filename[0]++;

remove(filename);

filename[0]--;

}

}

}

FileSort::FileSort(int file\_count) {

this->file\_count = file\_count;

groupSrc = new fstream[file\_count];

groupDst = new fstream[file\_count];

}

FileSort::~FileSort() {

delete[] groupSrc;

delete[] groupDst;

}

**sort.cpp**

#include "LAB7.h"

bool FileSort::sort\_A01(int& iteration\_counter) {

modeA = true;

open\_files(file\_count, true);

iteration\_counter = -1;//Начинаем с распределения из исходного файла, не является проходом

while (true) {

++iteration\_counter;

sort\_iteration();

for (int i = 0; i < file\_count; i++) {

groupSrc[i].clear();

groupSrc[i].seekg(0, ios::beg);

groupDst[i].clear();

groupDst[i].seekg(0, ios::beg);

}

if (check\_eof(groupDst[1])) break;

modeA = !modeA;

//Закрытие файлов

for (int i = 0; i < file\_count; i++) {

if (groupDst[i].is\_open()) groupDst[i].close();

if (groupSrc[i].is\_open()) groupSrc[i].close();

}

//Открытие файлов с переключённым режимом modeA

open\_files(file\_count);

}

clear\_files(file\_count);

return !modeA;

}

void FileSort::sort\_iteration() {

//Счётчик полностью прочитанных Src

int eof\_count = 0;

//Массив полностью прочитанных Src

bool\* eof = new bool[file\_count];

for (int i = 0; i < file\_count; i++)

eof[i] = false;

//Подготовить первые элементы всех Src

Value\* value\_current = new Value[file\_count];

for (int i = 0; i < file\_count; i++)

if (check\_eof(groupSrc[i])) {

eof[i] = true;

eof\_count++;

value\_current[i] = INT\_MIN;

}

else

value\_current[i] = Value::read(groupSrc[i]);

//Счётчик рассматриваемых Src для текущей последовательности

int active\_count = 0;

//Массив рассматриваемых Src для текущей последовательности

bool\* active = new bool[file\_count];

for (int i = 0; i < file\_count; i++)

active\_count += active[i] = !eof[i];

//Основной цикл сортировки

int current\_dst = 0;

while (eof\_count < file\_count) {

//Индекс Src с которого берём элемент

int ins = -1;

for (int i = 0; i < file\_count; i++) {

if (active[i] && (value\_current[i].key < value\_current[ins].key || ins == -1))

ins = i;

}

//Записываем элемент в Dst

value\_current[ins].write(groupDst[current\_dst]);

//Смотрим закончился ли файл или готовим следующий элемент на очередь

if (check\_eof(groupSrc[ins])) {

eof\_count++;

active\_count--;

eof[ins] = true;

active[ins] = false;

value\_current[ins].key = INT\_MIN;

}

else {

//Подготавливаем элемент, если он не начинает новую последовательность (тогда будет < предыдущего)

Value value = Value::read(groupSrc[ins]);

if (value.key < value\_current[ins].key) {

active[ins] = false;

active\_count--;

}

value\_current[ins] = value;

}

//Если отсортировали по одной последовательности от всех Src

//в одну последовательность текущего Dst,

//то переход к очередному Dst

if (!active\_count) {

active\_count = 0;

for (int i = 0; i < file\_count; i++) {

active\_count += ((active[i] = !check\_eof(groupSrc[i]) || value\_current[i].key != INT\_MIN));

}

eof\_count = file\_count - active\_count;

current\_dst = (current\_dst + 1) % file\_count;

}

}

/\* Результат. Файлы ИЗ и Файлы В\*/

cout << "{";

for (int i = 0; i < file\_count; i++) {

if (groupSrc[i].is\_open()) {

cout << '\n' << i << ":\n";

cout << " <" << read\_fs(groupSrc[i]) << endl;

cout << " ----[" << read\_fs(groupDst[i]);

}

}

cout << "\n}\n";

//\*/

delete[] value\_current;

delete[] active;

delete[] eof;

}

**table.cpp**

#include "LAB7.h"

void Table::init\_draw() {

drawline\_step = 0;

min\_val = data[0];

max\_val = data[0];

for (int i = 1; i < length; i++) {

min\_val = min(min\_val, data[i]);

max\_val = max(max\_val, data[i]);

}

mcol = (int)round(max\_val);

}

bool Table::draw\_line(string& line) {

line = "";

if (drawline\_step == -1) return false;

if (drawline\_step == 0) {

string nm;

for (int i = 0; i < length \* column\_width; i++) {

if(i%column\_width == 0) nm = to\_string(i/column\_width + 2);

line += (column\_width - i % column\_width <= nm.size()) ? nm[nm.size() - (column\_width - i % column\_width)] : ' ';

}

for (int i = 0; i < legendr1.size() + to\_string(mcol).size() + 4; i++) line += " ";

}

else if (drawline\_step == 1) {

line += " ";

for (int i = 0; i < length \* column\_width; i++) line += '-';

for (int i = 0; i < legendr1.size() + to\_string(mcol).size() + 3; i++) line += " ";

}

else if (drawline\_step - 2 < display\_h) {

int row = drawline\_step - 2;

int dif = max\_val - min\_val;

line += '|';

for (int i = 0; i < length; i++) {

int val = data[i]-min\_val;

for (int w = 0; w < column\_width; w++) {

if (!dif) line += (char)178;

else line +=

(val >= (display\_h - row) \* (dif / (float)display\_h)) ? (char)178 :

(val && val >= (display\_h - row - 1) \* (dif / (float)display\_h)) ? (char)177 :

' ';

}

}

line += "|";

int col = (int)round((display\_h - row) \* (dif / (float)display\_h));

if (row == 0) line += "\033[107m\033[30m\033[4m";

line += to\_string(min\_val+col);

if (row == 0) line += "\033[49m\033[0m\033[24m";

if (row == 0) {

line += " " + legendr1 + " ";

}

else {

for (int i = 0; i < legendr1.size() + to\_string(mcol).size() - to\_string(col+min\_val).size() + 2; i++) line += " ";

}

}

else {

line += " ";

for (int i = 0; i < length \* column\_width; i++) line += '-';

for (int i = 0; i < legendr1.size() + to\_string(mcol).size() + 3; i++) line += " ";

drawline\_step = -1;

return true;

}

drawline\_step++;

return true;

}

Table::operator string() {

string result = "";

string s;

init\_draw();

while (draw\_line(s)) {

result += s + '\n';

}

return result;

};