МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**Лабораторная работа № 2**

по курсу «Объектно-ориентированное программирование»

**«Наследование в C++»**

**15 ВАРИАНТ**

Выполнил:

студенты гр. КТбо2-1

Самардак А.В.

Проверил:

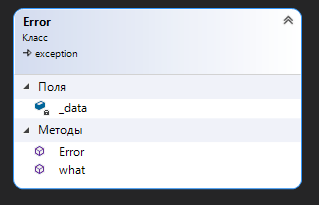
Тарасов С.А.

**Таганрог 2020**

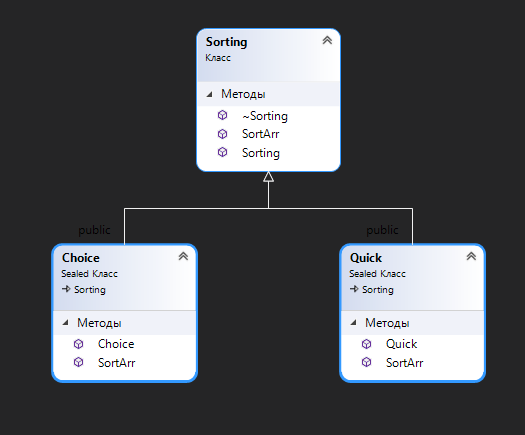
1. **Вариант задания №15**

Создать класс Sorting(сортировка), и на его основе классы Choice(сортировка выбором) и Quick(быстрая сорти-ровка). Размер сортируемых массивов задается при их создании, а элементы считываются из файла.

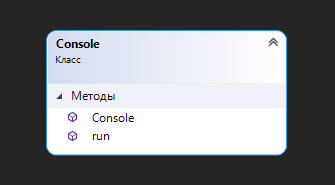
1. **UML-диаграмма класса Error**



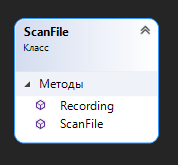
1. **UML-диаграмма классов: Sorting, Quick, Choice**



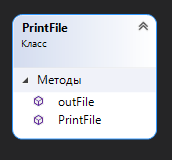
1. **UML-диаграмма класса Console**



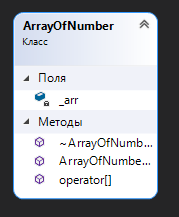
1. **UML-диаграмма класса ScanFile**



1. **UML-диаграмма класса PrintFile**



1. **UML-диаграмма класса ArrayOfNumber**



**main.cpp:**

#include "Console.h"

#include <vld.h>

int main() {

Console console;

console.run();

return 0;

}

**Error.h:**

#pragma once

#include <string>

class Error: public std::exception {

public:

Error(const std::string& value);

const char\* what() const override;

private:

std::string \_data;

};

**Error.cpp:**

#include "Error.h"

Error::Error(const std::string& value): \_data(value) {}

const char\* Error::what() const {

return \_data.c\_str();

}

**Console.h:**

#pragma once

#include <iostream>

#include <clocale>

#include "Sorting.h"

#include "Choice.h"

#include "Quick.h"

#include "ScanFile.h"

#include "PrintFile.h"

#include "ArrayOfNumber.h"

#include "Error.h"

class Console {

public:

Console();

void run();

};

**Console.cpp:**

#include "Console.h"

Console::Console() = default;

void Console::run() {

setlocale(LC\_ALL, "Russian");

std::string name\_file;

int check, size;

do {

std::cout << "Какую сортировку выбрать?:\n 0. Choice\n 1. Quick" << std::endl;

std::cin >> check;

} while (check < 0 || check > 1);

std::cout << "Название файла: " << std::endl;

std::cin >> name\_file;

std::cout << "Сколько элементов считать с файла?: " << std::endl;

std::cin >> size;

try {

ArrayOfNumber arr(size);

ScanFile scan\_file;

scan\_file.Recording(arr, size, name\_file);

PrintFile print\_file;

Sorting\* sort = nullptr;

if (!check) {

sort = new Choice;

} else {

sort = new Quick;

}

sort->SortArr(arr, size);

print\_file.outFile(arr, size);

delete sort;

} catch (const std::exception& exept) {

std::cout << exept.what() << std::endl;

}

}

**Sorting.h:**

#pragma once

#include "ArrayOfNumber.h"

#include <algorithm>

class Sorting {

public:

Sorting();

virtual ~Sorting();

virtual void SortArr(const ArrayOfNumber& arr, int size) const = 0;

};

**Sorting.cpp:**

#include "Sorting.h"

Sorting::Sorting() = default;

Sorting::~Sorting() {}

**Choice.h:**

#pragma once

#include "Sorting.h"

class Choice final : public Sorting {

public:

Choice();

void SortArr(const ArrayOfNumber& arr, int size) const;

};

**Choice.cpp:**

#include "Choice.h"

Choice::Choice() = default;

void Choice::SortArr(const ArrayOfNumber& arr, int size) const {

for (int i = 0; i < size - 1; ++i) {

int i\_min = i;

for (int j = i + 1; j < size; ++j) {

if (arr[j] < arr[i\_min]) {

i\_min = j;

}

}

std::swap(arr[i], arr[i\_min]);

}

}

**Quick.h:**

#pragma once

#include "Sorting.h"

class Quick final : public Sorting {

public:

Quick();

void SortArr(const ArrayOfNumber& arr, int size) const;

};

**Quick.cpp:**

#include "Quick.h"

Quick::Quick() = default;

void Quick::SortArr(const ArrayOfNumber& arr, int size) const {

int i = 0;

int j = size - 1;

int mid = arr[size / 2];

do {

while (arr[i] < mid) {

i++;

}

while (arr[j] > mid) {

j--;

}

if (i <= j) {

std::swap(arr[i], arr[j]);

i++;

j--;

}

} while (i <= j);

if (j > 0) {

SortArr(arr, j + 1);

}

if (i < size) {

SortArr(arr, size - i);

}

}

**ScanFile.h:**

#pragma once

#include "ArrayOfNumber.h"

#include "Error.h"

#include <fstream>

class ScanFile {

public:

ScanFile();

void Recording(const ArrayOfNumber& arr, int size, const std::string& name\_file) const;

};

**ScanFile.cpp:**

#include "ScanFile.h"

ScanFile::ScanFile() = default;

void ScanFile::Recording(const ArrayOfNumber& arr, int size, const std::string& name\_file) const {

std::ifstream fin(name\_file);

if (!fin.is\_open()) {

throw Error("Нет такого файла...");

}

else {

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

if (!(fin >> arr[i])) {

throw Error("Недопустимое количество элементов...");

}

}

}

fin.close();

}

**PrintFile.h:**

#pragma once

#include <fstream>

#include "ArrayOfNumber.h"

class PrintFile {

public:

PrintFile();

void outFile(const ArrayOfNumber& arr, int size) const;

};

**PrintFile.cpp:**

#include "PrintFile.h"

PrintFile::PrintFile() = default;

void PrintFile::outFile(const ArrayOfNumber& arr, int size) const {

std::ofstream fout("sort\_array.txt");

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

fout << arr[i] << "\n";

}

fout.close();

}

**ArrayOfNumber.h:**

#pragma once

class ArrayOfNumber {

public:

ArrayOfNumber();

ArrayOfNumber(int size);

~ArrayOfNumber();

int& operator [] (int i) const;

private:

int\* \_arr;

};

**ArrayOfNumber.cpp:**

#include "ArrayOfNumber.h"

ArrayOfNumber::ArrayOfNumber(): \_arr(new int[1000]) {}

ArrayOfNumber::ArrayOfNumber(int size): \_arr(new int[size]) {}

ArrayOfNumber::~ArrayOfNumber() {

delete[] \_arr;

}

int& ArrayOfNumber::operator [] (int i) const {

return \_arr[i];

}