

Звіт

Автор: Богданов І.Ю. КІТ-119а Дата: 16 квітня 2020

Лабораторна робота №5. Агрегація та композиція

Тема. Класи. Агрегація. Композиція. Ключові слова *typedef* та *auto*.

Мета: отримати поняття агрегація та композиція; отримати знання про призначення ключових слів *typedef* та *auto*.

1. Завдання до роботи **Індивідуальне завдання:**

Змінити попередню лабораторну роботу й продемонструвати агрегацію і композицію.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: `file`

Клас, що має в собі масив базового класу та методи для роботи з ним: `dir`

Клас, що повинен демонструвати агрегацію: `x_version`

Клас, що повинен демонструвати композицію: `options`

2.2 Опис змінних

`int ver` – поле класу `x_version`(64 або 32 розрядна програма).

`int size` – поле класу `file`(розмір файлу у бітах).

`int index` – поле класу `file`(унікальний індекс).

`bool is_system` – поле класу `options`(чи є файл системним).

`bool is_hidden` – поле класу `options`(чи є файл прихованим).

`std::string name` – поле класу `file`(назва файлу).

`x_version* x` – поле класу `file`(64 або 32 розрядна програма).

`options opt` – поле класу `file`(чи є файл прихованим і системним).

`int next_ind` – поле класу `dir`(номер наступного файлу у директорії).

`int new_ind` – поле класу `dir`(індекс наступного файлу у директорії).

`file*` `files` – поле класу `dir`(масив елементів класу `file`).

`file*` `copy` – поле класу `dir`(показчик на клас `file`, використовується для правильної роботи деяких методів).

2.3 Опис методів

`int` `get_ver()` `const` – отримання значення поля `ver` змінної класу `x_version`(метод класу `x_version`).

`void` `change_ver(const int&)` – зміна значення поля `ver` змінної класу `x_version`(метод класу `x_version`).

`int` `get_x()` `const` – отримання значення поля `x` змінної класу `file`(метод класу `file`).

`int` `get_size()` `const` – отримання значення поля `size` змінної класу `file`(метод класу `file`).

`int` `get_index()` `const` – отримання значення поля `index` змінної класу `file`(метод класу `file`).

`bool` `get_sys()` `const` – отримання значення поля `is_system` змінної класу `file`(метод класу `file`).

`bool` `get_hid()` `const` – отримання значення поля `is_hidden` змінної класу `file`(метод класу `file`).

`std::string` `get_name()` `const` – отримання значення поля `name` змінної класу `file`(метод класу `file`).

`void` `change_x(const int &x)` – зміна значення поля `x` змінної класу `file`(метод класу `file`).

`void` `change_size(const int &sz)` – зміна значення поля `size` змінної класу `file`(метод класу `file`).

`void` `change_index(const int &in)` – зміна значення поля `index` змінної класу `file`(метод класу `file`).

`void` `change_sys(const bool&)` – зміна значення поля `is_system` змінної класу `file`(метод класу `file`).

`void` `change_hid(const bool&)` – зміна значення поля `is_hidden` змінної класу `file`(метод класу `file`).

`void` `change_name(const std::string&)` – зміна значення поля `name` змінної класу `file`(метод класу `file`).

`file()` – конструктор класу `file`.

`file(const file&)` – конструктор копіювання класу `file`.

`file(const int&, const int&, const int&, const bool&, const bool&, const std::string&)` – конструктор з параметрами класу `file`.

`~file()` – деструктор класу `file`.

`void` `add_file(const file &f)` – додавання об'єкту класу `file` до масиву в класі `dir`(метод класу `dir`).

`void del_file(const int &index)` – видалення об'єкту класу `file` з масиву в класі `dir` (метод класу `dir`).

`void del_all()` – видалення усіх об'єктів класу `file` з масиву в класі `dir` (метод класу `dir`).

`void add_file_by_str(const std::string& str, x_version*, x_version*)` – додавання об'єкту класу `file` до масиву в класі `dir` за допомогою строки з інформацією про об'єкт (метод класу `dir`).

`void read_from_file(const std::string& name, x_version*, x_version*)` – заповнення масиву об'єктів класу `file` інформація про які буде зчитана з файлу (метод класу `dir`).

`file get_file_by_index(const int& index) const` – отримання об'єкту класу `file` з масиву в класі `dir` (метод класу `dir`).

`void get_file_to_screen(const int &index) const` – виведення об'єкту класу `file` з масиву в класі `dir` на екран (метод класу `dir`).

`void print_all() const` – виведення усіх об'єктів класу `file` з масиву в класі `dir` на екран (метод класу `dir`).

`int count_system() const` – розрахування кількості скритих і системних файлів в об'єкті класу `dir` (метод класу `dir`).

`void print_to_file(const std::string& name) const` – запис у файл інформації про об'єкти класу `file` що є в масиві (метод класу `dir`).

`void get_file_to_screen(const int &index) const` – запис у рядок інформації про об'єкт класу `file` (метод класу `dir`).

`bool check_str(const std::string& str) const` – перевірка рядка на відповідність формату зберігання даних про об'єкт класу `file` (метод класу `dir`).

`void print_all_with_2_or_more_words() const` – виведення усіх об'єктів класу `file` в назві яких є 2 або більше слів з масиву в класі `dir` на екран (метод класу `dir`).

`void sort_files(bool (*f)(file&, file&))` – сортування усіх об'єктів класу `file` в об'єкті класу `dir` на екран (метод класу `dir`).

`bool get_sys() const` – отримання значення поля `is_system` змінної класу `options` (метод класу `options`).

`bool get_hid() const` – отримання значення поля `is_hidden` змінної класу `options` (метод класу `options`).

`void change_sys(const bool&)` – зміна значення поля `is_system` змінної класу `options` (метод класу `options`).

`void change_hid(const bool&)` – зміна значення поля `is_hidden` змінної класу `options` (метод класу `options`).

2.4 Опис функцій

`void menu()` – функція меню.

`bool sort_version(file&, file&)` – функція порівняння двох файлів.

`bool sort_size(file&, file&)` – функція порівняння двох файлів.

`bool sort_ind(file&, file&)` – функція порівняння двох файлів.

`bool` sort_opt(`file&`, `file&`)– функція порівняння двох файлів.
`bool` sort_name(`file&`, `file&`)– функція порівняння двох файлів.
`bool` sort_version_2(`file&`, `file&`)– функція порівняння двох файлів.
`bool` sort_size_2(`file&`, `file&`)– функція порівняння двох файлів.
`bool` sort_ind_2(`file&`, `file&`)– функція порівняння двох файлів.
`bool` sort_opt_2(`file&`, `file&`)– функція порівняння двох файлів.
`bool` sort_name_2(`file&`, `file&`)– функція порівняння двох файлів.

3 Текст програми

Лабораторная работа 5.cpp

```
#include <iostream>
#include "menu.h"
#define _CRTDBG_MAP_ALLOC

int main()
{
    menu();
    if (_CrtDumpMemoryLeaks()) {
        std::cout << "Утечка памяти обнаружена." << "\n";
    }
    else {
        std::cout << "Утечка памяти не обнаружена." << "\n";
    }
}

file.h
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <regex>
#include <cstdint>

typedef size_t size_t_t;

class x_version {
private:
    int ver;
public:
    int get_ver() const;
    void change_ver(const int&);
};

class options {
private:
    bool issystem;
    bool ishidden;
public:
    bool get_sys() const;
    bool get_hid() const;
    void change_sys(const bool&);
    void change_hid(const bool&);
};

class file {
private:
    x_version* x;
    size_t_t size;
    size_t_t index;
    options opt;
    std::string name;
public:
    size_t_t get_x() const;
    size_t_t get_size() const;
    size_t_t get_index() const;
    bool get_sys() const;
```

```

    bool get_hid() const;
    std::string get_name() const;
    void change_x(x_version*);
    void change_size(const size_t_t&);
    void change_index(const size_t_t&);
    void change_sys(const bool&);
    void change_hid(const bool&);
    void change_name(const std::string&);
    file();
    file(const file&);
    file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const
std::string&);
    ~file();
};
file.cpp
#include "file.h"

size_t_t file::get_x() const {
    return x->get_ver();
}
size_t_t file::get_size() const {
    return size;
}
size_t_t file::get_index() const {
    return index;
}
bool file::get_sys() const {
    return opt.get_sys();
}
bool file::get_hid() const {
    return opt.get_hid();
}
std::string file::get_name() const {
    return name;
}
void file::change_x(x_version* new_x) {
    x = new_x;
}
void file::change_size(const size_t_t& sz) {
    size = sz;
}
void file::change_index(const size_t_t& in) {
    index = in;
}
void file::change_sys(const bool &sys) {
    opt.change_sys(sys);
}
void file::change_hid(const bool &hid) {
    opt.change_hid(hid);
}
void file::change_name(const std::string &nm) {
    name = nm;
}
file::file() {
    x = 0;
    size = 100;
    index = 0;
    opt.change_hid(false);
    opt.change_sys(false);
    name = "File";
    std::cout << "Файл создан при помощи конструктора поумолчанию." << "\n";
}
file::file(const file &f) {
    x = f.x;
    size = f.size;
    index = f.index;
    opt = f.opt;
    name = f.name;
}

```

```

file::file(x_version* ver, const size_t_t &sz, const size_t_t &ind, const bool &sys, const bool
&hid, const std::string &nm) {
    x = ver;
    size = sz;
    index = ind;
    opt.change_sys(sys);
    opt.change_hid(hid);
    name = nm;
    std::cout << "Файл создан при помощи конструктора с аргументами." << "\n";
}
file::~file() {
    std::cout << "Файл уничтожен при помощи деструктора поумолчанию." << "\n";
}
int x_version::get_ver() const {
    return ver;
}
void x_version::change_ver(const int& x) {
    ver = x;
}
bool options::get_sys() const {
    return issystem;
}
bool options::get_hid() const {
    return ishidden;
}
void options::change_sys(const bool& sys) {
    issystem = sys;
}
void options::change_hid(const bool& hid) {
    ishidden = hid;
}
dir.h
#pragma once
#include "file.h"

class dir {
private:
    file* files;
    file* copy;
    int next_ind = 0;
    int new_ind = 1;
public:
    void add_file(const file &f);
    void del_file(const int &index);
    void del_all();
    void add_file_by_str(const std::string& str, x_version*, x_version*);
    void read_from_file(const std::string& name, x_version*, x_version*);
    void get_file_to_screen(const int &index) const;
    std::string get_file_to_str(const int& index) const;
    file get_file_by_index(const int& index) const;
    void print_all() const;
    int count_system() const;
    void print_to_file(const std::string& name) const;
    bool check_str(const std::string& str) const;
    void print_all_with_2_or_more_words() const;
    void sort_files(bool (*f)(file&,file&));
};

bool sort_version(file&, file&);
bool sort_size(file&, file&);
bool sort_ind(file&, file&);
bool sort_opt(file&, file&);
bool sort_name(file&, file&);
bool sort_version_2(file&, file&);
bool sort_size_2(file&, file&);
bool sort_ind_2(file&, file&);
bool sort_opt_2(file&, file&);
bool sort_name_2(file&, file&);
dir.cpp

```

```

#include "dir.h"

void dir::add_file(const file &f) {
    if (next_ind == 0) {
        files = new file[1];
        files[next_ind] = f;
        files[next_ind].change_index(new_ind);
        new_ind++;
        next_ind++;
    }
    else {
        copy = new file[next_ind];
        for (size_t i = 0; i < next_ind; i++) {
            copy[i] = files[i];
        }
        delete[] files;
        files = new file[next_ind + 1];
        for (size_t i = 0; i < next_ind; i++) {
            files[i] = copy[i];
        }
        delete[] copy;
        files[next_ind] = f;
        files[next_ind].change_index(new_ind);
        next_ind++;
        new_ind++;
    }
}

void dir::del_file(const int &index) {
    if (next_ind == 1) {
        delete[] files;
        next_ind--;
    }
    else {
        copy = new file[next_ind - 1];
        for (int i = 0; i < index; i++) {
            copy[i] = files[i];
        }
        for (int i = index, j = index + 1; i < (next_ind - 1), j < next_ind; i++, j++) {
            copy[i] = files[j];
        }
        delete[] files;
        files = new file[next_ind - 1];
        for (int i = 0; i < next_ind - 1; i++) {
            files[i] = copy[i];
        }
        delete[] copy;
        next_ind--;
    }
}

void dir::del_all() {
    if (next_ind != 0) {
        delete[] files;
        next_ind = 0;
    }
}

void dir::read_from_file(const std::string& name, x_version* x32, x_version* x64) {
    del_all();
    std::ifstream f;
    char* temp;
    f.open(name);
    while (!f.eof()) {
        temp = new char[100];
        f.getline(temp, 100);
        add_file_by_str(temp, x32, x64);
        delete[] temp;
    }
    f.close();
}

std::string dir::get_file_to_str(const int &index) const {

```

```

        std::stringstream s;
        s << "\"" << files[index].get_name() << "\" " << files[index].get_index() << " " <<
files[index].get_size() << " " << files[index].get_x() << " " << files[index].get_hid() << " " <<
files[index].get_sys();
        return s.str();
    }
}

void dir::print_all() const {
    for (int i = 0; i < next_ind; i++) {
        std::cout << i + 1 << " ";
        std::string str;
        str = get_file_to_str(i);
        std::cout << str << "\n";
    }
}

int dir::count_system() const {
    auto count = 0;
    for (int i = 0; i < next_ind; i++) {
        if (files[i].get_hid() && files[i].get_sys()) {
            count++;
        }
    }
    return count;
}

file dir::get_file_by_index(const int& index) const {
    for (int i = 0; i < next_ind; i++) {
        if (files[i].get_index() == index) {
            return files[i];
        }
    }
}

}

void dir::add_file_by_str(const std::string& str, x_version* x32, x_version* x64) {
    if (check_str(str)) {
        std::regex re("\\.+\\");
        std::smatch sm;
        std::regex_search(str, sm, re);
        auto i = str.find("\\");
        i = str.find("\\", i + 1);
        std::regex re_temp("\\");
        std::string temp = sm[0];
        std::string name = std::regex_replace(temp, re_temp, "");
        auto i2 = str.find(" ", i + 2);
        temp = str.substr(i + 1, i2 - i);
        std::stringstream s;
        s << temp;
        auto index = 0;
        s >> index;
        int i3 = str.find(" ", i2 + 1);
        s.clear();
        temp = str.substr(i2 + 1, i3 - i2);
        s << temp;
        auto size = 0;
        s >> size;
        auto i4 = str.find(" ", i3 + 1);
        s.clear();
        temp = str.substr(i3 + 1, i4 - i3);
        s << temp;
        auto x = 0;
        s >> x;
        x_version* temp_x = 0;
        if (x == 32) {
            temp_x = x32;
        }
        else if (x == 64) {
            temp_x = x64;
        }
        auto i5 = str.find(" ", i4 + 1);
        s.clear();
        temp = str.substr(i4 + 1, i5 - i4);
        s << temp;
    }
}

```



```

        auto hid = false;
        s >> hid;
        auto i6 = str.find(" ", i5 + 1);
        s.clear();
        temp = str.substr(i5 + 1, i6 - i5);
        s << temp;
        auto sys = false;
        s >> sys;
        file new_file(temp_x, size, index, sys, hid, name);
        add_file(new_file);
    }
}

void dir::print_to_file(const std::string& name) const {
    std::ofstream f;
    f.open(name);
    std::string str;
    for (int i = 0; i < next_ind; i++) {
        str = get_file_to_str(i);
        f << str;
        if (i != next_ind - 1) {
            f << "\n";
        }
    }
    f.close();
}

bool dir::check_str(const std::string& str) const {
    std::regex re("[A-Za-zA-Яa-я0-9\\s\\!,\\?\\\"\\.\\.;\\' ]*");
    if (!(std::regex_search(str, re))) {
        return false;
    }
    std::regex re_2("\\s{2,}");
    if (std::regex_search(str, re_2)) {
        return false;
    }
    std::regex re_3("[\\!\\?:\\.\\,\\;]{2,}");
    if (std::regex_search(str, re_3)) {
        return false;
    }
    std::regex re_4("[\\'\\\" ]{2,}");
    if (std::regex_search(str, re_4)) {
        return false;
    }
    std::regex re_5("^\"[A-ZА-Я]");
    if (std::regex_search(str, re_5)) {
        return false;
    }
    return true;
}

void dir::print_all_with_2_or_more_words() const {
    for (int i = 0; i < next_ind; i++) {
        std::string str;
        str = get_file_to_str(i);
        std::regex re("\\\".+\\.+\\\"");
        if (std::regex_search(str, re)) {
            std::cout << i + 1 << " " << str << "\n";
        }
    }
}

void dir::sort_files(bool (*f)(file&, file&)) {
    bool check = false;
    file temp;
    do {
        check = false;
        for (int i = 0; i < next_ind - 1; i++) {
            if (f(files[i], files[i + 1])) {
                temp = files[i];
                files[i] = files[i + 1];
                files[i + 1] = temp;
                check = true;
            }
        }
    } while (check);
}

```

```

        }
    } while (check);
}
bool sort_version(file& first, file& second) {
    return (first.get_x() < second.get_x());
}
bool sort_size(file& first, file& second) {
    return (first.get_size() < second.get_size());
}
bool sort_ind(file& first, file& second) {
    return (first.get_index() > second.get_index());
}
bool sort_opt(file& first, file& second) {
    if (((second.get_sys() && second.get_hid()) == true) && ((first.get_sys() && first.get_hid())
!= true)) {
        return true;
    }
    else if (((second.get_sys() == false) && (second.get_hid() == true)) && (first.get_hid() !=
true)) {
        return true;
    }
    else if (((second.get_hid() == false) && (first.get_hid() == false)) && ((second.get_sys() ==
true) && (first.get_sys() != true))) {
        return true;
    }
    else {
        return false;
    }
}
bool sort_name(file& first, file& second) {
    return (first.get_name() > second.get_name());
}
bool sort_version_2(file& first, file& second) {
    return (first.get_x() > second.get_x());
}
bool sort_size_2(file& first, file& second) {
    return (first.get_size() > second.get_size());
}
bool sort_ind_2(file& first, file& second) {
    return (first.get_index() < second.get_index());
}
bool sort_opt_2(file& first, file& second) {
    if (((second.get_sys() && second.get_hid()) == true) && ((first.get_sys() && first.get_hid())
!= true)) {
        return false;
    }
    else if (((second.get_sys() == false) && (second.get_hid() == true)) && (first.get_hid() !=
true)) {
        return false;
    }
    else if (((second.get_hid() == false) && (first.get_hid() == false)) && ((second.get_sys() ==
true) && (first.get_sys() != true))) {
        return false;
    }
    else {
        return true;
    }
}
bool sort_name_2(file& first, file& second) {
    return (first.get_name() < second.get_name());
}
}
menu.h
#pragma once
#include "dir.h"

void menu();
menu.cpp
#include "menu.h"

```

```

void menu() {
    setlocale(LC_ALL, "Russian");
    int n = 0, temp_i;
    dir directory;
    x_version* x32 = new x_version;
    x32->change_ver(32);
    x_version* x64 = new x_version;
    x64->change_ver(64);
    directory.read_from_file("data.txt",x32,x64);
    int s;
    int m;
    while (n != 4) {
        std::cout << "Выберите желаемую опцию:" << "\n";
        std::cout << "1 - добавить элемент в список." << "\n";
        std::cout << "2 - удалить элемент из списка." << "\n";
        std::cout << "3 - показать все элементы списка." << "\n";
        std::cout << "4 - завершить работу программы." << "\n";
        std::cout << "5 - посчитать количество скрытых системных файлов." << "\n";
        std::cout << "6 - прочитать данные из файла. " << "\n";
        std::cout << "7 - записать текущий список данных в файл. " << "\n";
        std::cout << "8 - найти все элеметы в названии которых есть 2 или больше слова. " << "\n";
        std::cout << "9 - отсортировать массив. " << "\n";
        std::cin >> n;
        if (n == 1) {
            directory.add_file_by_str("\File\ 0 123 64 0 0 \0",x32,x64);
            std::cout << "Файл добавлен." << "\n";
        }
        else if (n == 2) {
            std::cout << "Введите номер удаляемого элемента (нумерация начинается с 1): ";
            std::cin >> temp_i;
            directory.del_file(temp_i - 1);
            std::cout << "Файл удалён. " << "\n";
        }
        else if (n == 3) {
            directory.print_all();
        }
        else if (n == 5) {
            std::cout << "Количество скрытых системных файлов: " << directory.count_system() <<
"\n";
        }
        else if (n == 6) {
            directory.read_from_file("data.txt",x32,x64);
        }
        else if (n == 7) {
            directory.print_to_file("data.txt");
        }
        else if (n == 8) {
            directory.print_all_with_2_or_more_words();
        }
        else if (n == 9) {
            std::cout << "Выберите порядок сортировки. 1 - по по алфавиту и от большого к меньшему,
2 - наоборот. " << "\n";
            std::cin >> m;
            std::cout << "Введите номер признака по которому хотите отсортировать массив: 1 - x, 2
- size, 3 - index, 4 - opt, 5 - name. " << "\n";
            std::cin >> s;
            if (s == 1 and m == 1) {
                directory.sort_files(sort_version);
            }
            else if (s == 1 and m == 2) {
                directory.sort_files(sort_version_2);
            }
            else if (s == 2 and m == 1) {
                directory.sort_files(sort_size);
            }
            else if (s == 2 and m == 2) {
                directory.sort_files(sort_size_2);
            }
        }
    }
}

```

```

        else if (s == 3 and m == 1) {
            directory.sort_files(sort_ind);
        }
        else if (s == 3 and m == 2) {
            directory.sort_files(sort_ind_2);
        }
        else if (s == 4 and m == 1) {
            directory.sort_files(sort_opt);
        }
        else if (s == 4 and m == 2) {
            directory.sort_files(sort_opt_2);
        }
        else if (s == 5 and m == 1) {
            directory.sort_files(sort_name);
        }
        else if (s == 5 and m == 2) {
            directory.sort_files(sort_name_2);
        }
    }
}
directory.del_all();
delete x32;
delete x64;
}
test.cpp
#include "menu.h"
#define _CRTDBG_MAP_ALLOC

int main() {
    setlocale(LC_ALL, "Russian");
    int n;
    x_version* x32 = new x_version;
    x_version* x64 = new x_version;
    x32->change_ver(32);
    x64->change_ver(64);
    file test_file_1(x32, 1, 1, true, true, "ABC");
    file test_file_2(x64, 2, 2, false, false, "BCD");
    std::cout << "Тест будет пройден если сейчас на экран будет выведена следующая
последовательность: 1 1 0 0 0 - ";
    std::cout << sort_version(test_file_1, test_file_2) << " " << sort_size(test_file_1,
test_file_2) << " " << sort_ind(test_file_1, test_file_2) << " " << sort_opt(test_file_1,
test_file_2) << " " << sort_name(test_file_1, test_file_2);
    std::cin >> n;
}
data.txt
"Sys file" 2 123 32 1 1
"Qwerty" 3 521 64 1 0
"Hello friend" 4 289 64 0 1
"Water" 5 10000 32 0 0

```

4. Результаты работы программы

Результаты работы программы:

```
Файл создан при помощи конструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл создан при помощи конструктора с аргументами.
Файл создан при помощи конструктора поумолчанию.
Файл создан при помощи конструктора поумолчанию.
Файл создан при помощи конструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл создан при помощи конструктора поумолчанию.
Файл создан при помощи конструктора поумолчанию.
Файл создан при помощи конструктора поумолчанию.
Файл создан при помощи конструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Выберите желаемую опцию:
1 - добавить элемент в список.
2 - удалить элемент из списка.
3 - показать все элементы списка.
4 - завершить работу программы.
5 - посчитать количество скрытых системных файлов.
6 - прочитайте данные из файла.
7 - записать текущий список данных в файл.
8 - найти все элементы в названии которых есть 2 или больше слова.
9 - отсортировать массив.
```

Результати тестів:

```
Файл создан при помощи конструктора с аргументами.
Файл создан при помощи конструктора с аргументами.
Тест будет пройден если сейчас на экран будет выведена следующая последовательность: 1 1 0 0 0 - 1 1 0 0 0
```

5. Висновки

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи з агрегацією та композицією.

Програма протестована, витоків пам'яті немає, виконується без помилок.