

МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ХП»

Кафедра «Обчислювальна техніка та програмування»

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.119а.

Розробники

Виконав:

студент групи КІТ-119а

_____/Богданов І.Ю./

Перевірив:

_____/аспірант Бартош М. В./

Харків 2020

ЗАТВЕРДЖЕНО

КІТ.119а.

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.119а.

Аркушів 27

Харків 2020

ЗМІСТ

Вступ.....	3
1 Поняття «Інформаційна система».....	3
1.1 Призначення та галузь застосування.....	3
1.2 Постановка завдання до розробки.....	3
2 Розробка інформаційно-довідкової системи.....	6
2.1 Розробка алгоритмів програми.....	6
2.1.1 Розробка методів класу <i>file</i>	6
2.1.2 Розробка методів класу <i>executable_file</i>	6
2.1.3 Розробка методів класу <i>non_executable_file</i>	6
2.1.4 Розробка методів класу <i>dir</i>	7
3 Схеми алгоритму програми.....	8
Висновок.....	9
Список джерел інформації.....	10
Додаток А. Текст програми.....	11
Додаток Б. Результати роботи програми	27

ВСТУП

Поняття «Інформаційна система»

Інформаційно-довідкові системи – це сукупність організаційних і технічних засобів, що призначені для керування базами даних і використовуються, наприклад, для ведення статистики, складання каталогів тощо. Вони полегшують оперування великими об'ємами професійно цінної інформації, виступаючи як засіб надійного збереження професійних знань, забезпечуючи зручний і швидкий пошук необхідних відомостей.

Призначення та галузь застосування

Призначення розробки – оперування даними про прикладну галузь література, а саме про підручники. Розроблена з використанням ієрархії класів програма дозволяє виконувати такі завдання: читання даних з файлу та їх запис у контейнер, запис даних з контейнера у файл, сортування елементів у контейнері за вказаними критеріями (поле та напрям задаються користувачем з клавіатури), виконання особистого завдання. Також було розроблено декілька інших класів, які слугують для: відображення діалогового меню, тестування розроблених методів класу, сортування.

Постановка завдання до розробки

В основі функціонування інформаційно-довідкових систем лежить обробка інформації. Режими її обробки можуть бути такими: пакетний, діалоговий, реального часу.

Пакетний режим визначає операції та їх послідовність з формування даних в ЕОМ і формування розрахунків безпосередньо на обчислювальному центрі чи відповідною системою.

Діалоговий режим забезпечує безпосередню взаємодію користувача з системою. Ініціатором діалогу може бути як користувач, так і ЕОМ. В останньому випадку на кожному кроці користувачу повідомляється, що треба робити.

Режим реального часу — режим обробки інформації системою при взаємодії з зовнішніми процесами в темпі ходу цих процесів.

В роботі буде реалізовано діалоговий режим обробки інформації, де ініціатором виступає ЕОМ.

Дані, що обробляються, в оперативній пам'яті можуть зберігатися у вигляді масиву або лінійного (одно- або двонаправленого) списку.

До переваг масиву можна віднести:

1. Ефективність при звертанні до довільного елемента, яке відбувається за постійний час $O(1)$,
2. Можливість компактного збереження послідовності їх елементів в локальній області пам'яті, що дозволяє ефективно виконувати операції з послідовного обходу елементів таких масивів.
3. Масиви є дуже економною щодо пам'яті структурою даних.

До недоліків:

1. Операції, такі як додавання та видалення елемента, потребують часу $O(n)$, де n — розмір масиву.
2. У випадках, коли розмір масиву є досить великий, використання звичайного звертання за індексом стає проблематичним.
3. Масиви переважно потребують неперервної області для зберігання.

До переваг списку можна віднести:

1. Списки досить ефективні щодо операцій додавання або видалення елемента в довільному місці списку, виконуючи їх за постійний час.
2. В списках також не існує проблеми «розширення», яка рано чи пізно виникає в масивах фіксованого розміру, коли виникає необхідність включити в нього додаткові елементи.
3. Функціонування списків можливо в ситуації, коли пам'ять комп'ютера фрагментована.

До недоліків:

1. Для доступу до довільного елементу необхідно пройти усі елементи перед ним.
2. Необхідність разом з корисною інформацією додаткового збереження інформації про вказівники, що позначається на ефективності використання пам'яті цими структурами.

Виходячи з переваг та недоліків зазначених вище в розроблюваній програмі для подання даних буде реалізовано вектор, який є абстрактною моделлю, що імітує динамічний масив.

Для реалізації поставленого завдання було обрано об'єктно-орієнтовану мову програмування C++, через те, що вона засновує програми як сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. А середовищем програмування – Microsoft Visual Studio.

РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ

Розробка алгоритмів програми

При розробленні структур даних було створено: базовий клас `file`, який наслідують класи `executable_file` та `non_executable_file`. На рис. 1 показано внутрішню структуру, а на рис. 2 - відносини розроблених класів у вигляді UML-діаграми.



Рисунок 1 – Поля базового класу (а), а також класів-спадкоємців (б, в)

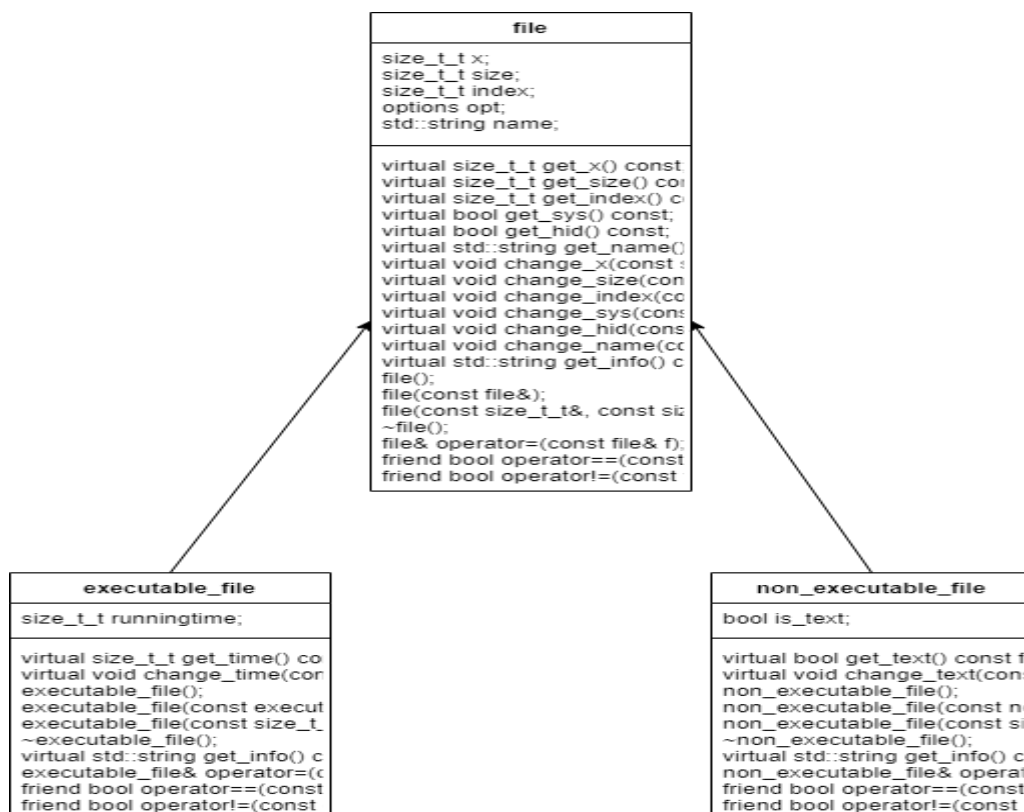


Рисунок 2 – Схема ієрархії розроблених класів

Дані про файли будуть заноситися до масиву. Для цього було розроблено клас-контролер dir з полями показаними на рис. 3 і методами на рис. 4.

Приватні
<pre>file** files; file** copy; int next_ind = 0; int new_ind = 1;</pre>

Рисунок 3 – Поля класу-контролеру dir

Методи
<pre>void add_file(const executable_file& f); void add_file(const non_executable_file& f); void del_file(const int& index); void del_all(); void add_file_by_str(const std::string& str); void read_from_file(const std::string& name); 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&)); file* operator[](const int& index); friend std::ostream& operator<<(std::ostream& on, dir& d);</pre>

Рисунок 4 – Розроблені методи класу dir

На рис. 5 подано структуру проекту розробленого програмного продукту.

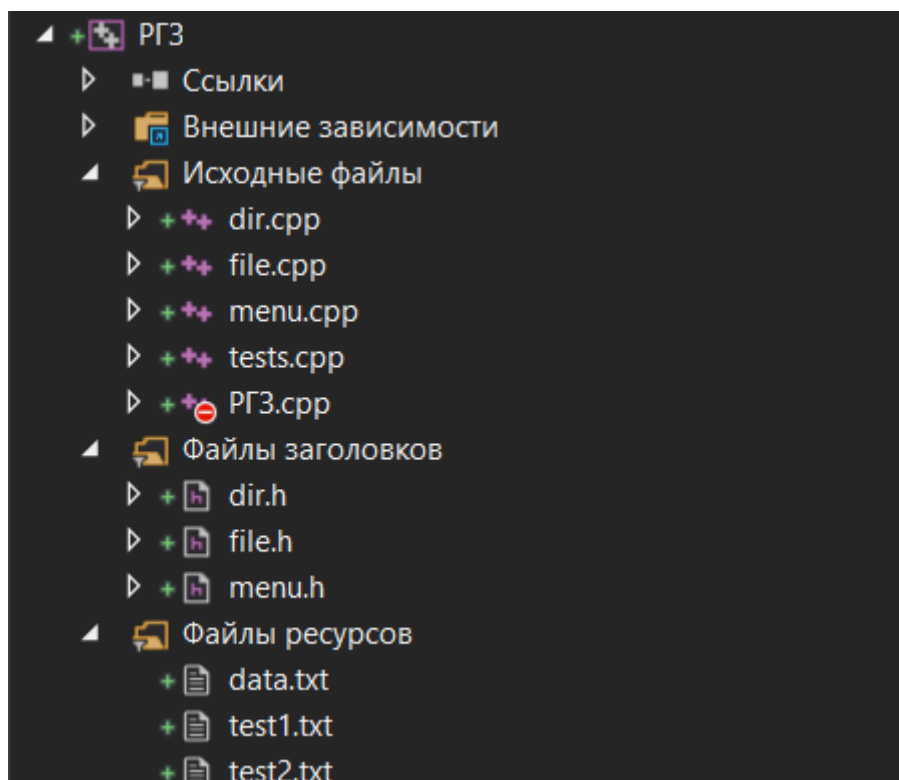


Рисунок 5 – Структура проекту

СХЕМИ АЛГОРИТМУ ПРОГРАМИ

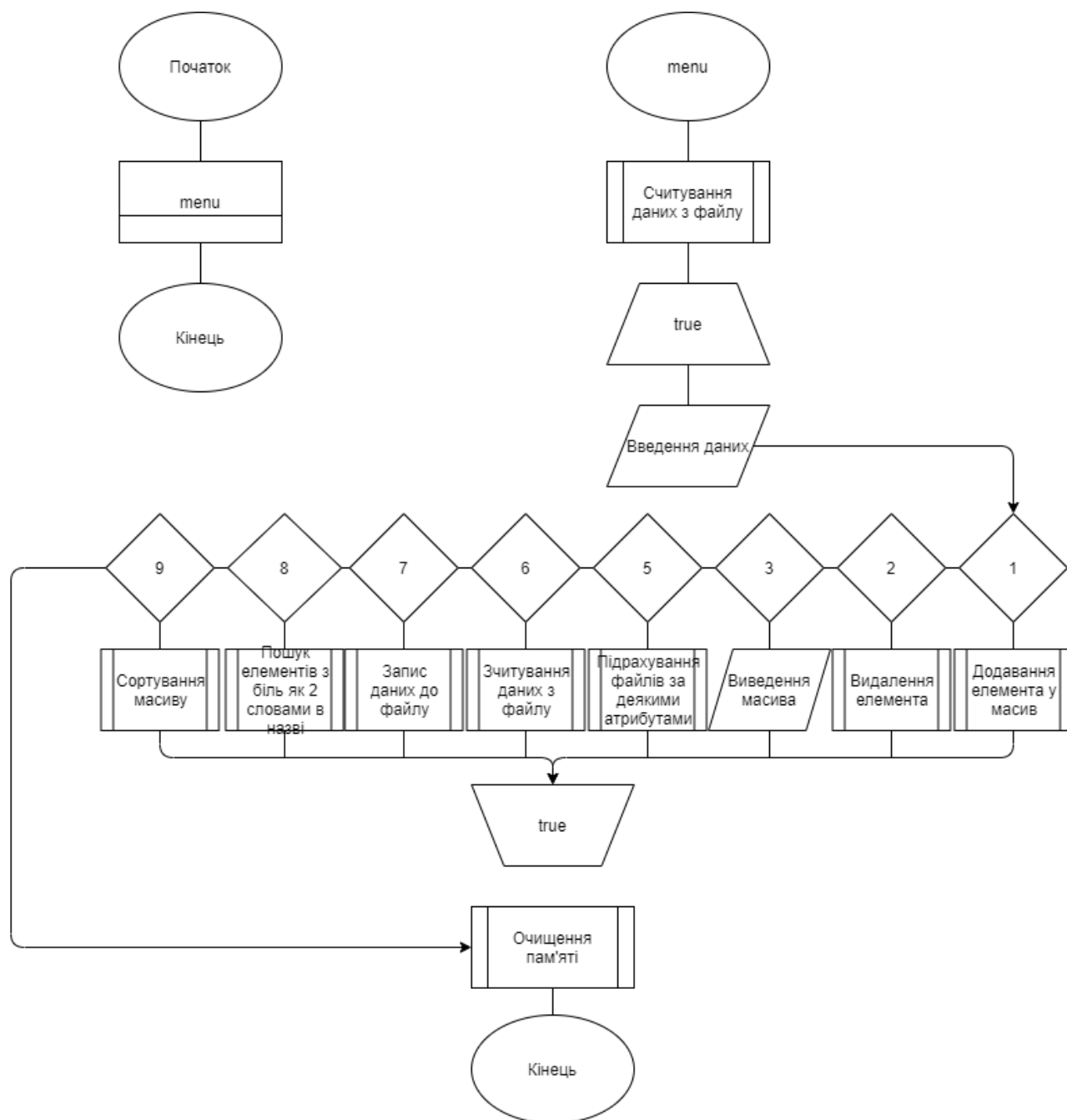


Рисунок 1 – Схема алгоритму функції menu

ВИСНОВОК

У результаті розробки інформаційно-довідкової системи було виконано наступні завдання:

1. Досліджено літературу стосовно прикладної галузі та оформлено аналітичний розділ пояснювальної записки;
2. Для прикладної галузі файл розроблено розгалужену ієрархію класів, що складається з трьох класів – один «батьківський», два спадкоємці. У них було перевантажено оператори введення-виведення та оператор порівняння;
3. Розроблено клас-контролер, що включає масив базового класу, та класів спадкоємців.
4. Оформлено схеми алгоритмів функцій класів контролера та діалогового меню;
5. Оформлено документацію;
6. Було додано обробку помилок, перевірку вхідних даних за допомогою регулярних виразів;

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Дейтел Х.М. Как программировать на Си++ / Х.М. Дейтел, П.Дж. Дейтел М. : ЗАО БИНОМ, 1999. – 1000 с.
2. Штейн Клифорд (2019). Алгоритмы. Построение и анализ.
3. Вандервуд, Джосаттис - Шаблоны С++. Справочник разработчика. / Пер. с англ. - М.: Вильямс, 2008. – 536 с.
4. Андрей Александреску, Современное проектирование на С++.М.:ООО «И.Д.Вильямс», 2002.
5. Страуструп Б. Дизайн и эволюция С++ / Б. Страуструп; пер. с англ. – М. : ДМК Пресс; С.Пб: Питер, 2007. – 445 с.
6. Остерн Обобщенное программирование и STL: Использование и наращивание стандартной библиотеки шаблонов С++ / Остерн; Пер. сангл. – С.Пб: Невский Диалект, 2004. – 544 с.

Додаток А

Текст програми

file.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <regex>
#include <iomanip>
#include <vector>
#include <list>
#include <set>
#include <map>

typedef size_t size_t_t;

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 {
protected:
    size_t_t x;
    size_t_t size;
    size_t_t index;
    options opt;
    std::string name;
public:
    int type_of_file = 0;
    virtual size_t_t get_x() const;
    virtual size_t_t get_size() const;
    virtual size_t_t get_index() const;
    virtual bool get_sys() const;
    virtual bool get_hid() const;
    virtual std::string get_name() const;
    virtual void change_x(const size_t_t&);
    virtual void change_size(const size_t_t&);
    virtual void change_index(const size_t_t&);
    virtual void change_sys(const bool&);
    virtual void change_hid(const bool&);
    virtual void change_name(const std::string&);
    virtual std::string get_info() const;
    file();
    file(const file&);
    file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const bool&,
const std::string&);
    ~file();
    file& operator=(const file& f);
    friend bool operator==(const file& f1, const file& f2);
    friend bool operator!=(const file& f1, const file& f2);
};

class executable_file final : public file {
private:
    size_t_t runningtime;
```

```

public:
    virtual size_t_t get_time() const final;
    virtual void change_time(const size_t_t&) final;
    executable_file();
    executable_file(const executable_file&);
    executable_file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const
bool&, const std::string&, const size_t_t&);
    ~executable_file();
    virtual std::string get_info() const final;
    executable_file& operator=(const executable_file& f);
    friend bool operator==(const executable_file& f1, const executable_file& f2);
    friend bool operator!=(const executable_file& f1, const executable_file& f2);
};
class non_executable_file final : public file {
private:
    bool is_text;
public:
    virtual bool get_text() const final;
    virtual void change_text(const bool&) final;
    non_executable_file();
    non_executable_file(const non_executable_file&);
    non_executable_file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&,
const bool&, const std::string&, const bool&);
    ~non_executable_file();
    virtual std::string get_info() const final;
    non_executable_file& operator=(const non_executable_file& f);
    friend bool operator==(const non_executable_file& f1, const non_executable_file& f2);
    friend bool operator!=(const non_executable_file& f1, const non_executable_file& f2);
};
bool check_str(const std::string& str);
bool operator==(const file& f1, const file& f2);
bool operator!=(const file& f1, const file& f2);
bool operator==(const executable_file& f1, const executable_file& f2);
bool operator!=(const executable_file& f1, const executable_file& f2);
bool operator==(const non_executable_file& f1, const non_executable_file& f2);
bool operator!=(const non_executable_file& f1, const non_executable_file& f2);
bool operator>(const file& f1, const file& f2);
bool operator<(const file& f1, const file& f2);
bool operator>(const executable_file& f1, const executable_file& f2);
bool operator<(const executable_file& f1, const executable_file& f2);
bool operator>(const non_executable_file& f1, const non_executable_file& f2);
bool operator<(const non_executable_file& f1, const non_executable_file& f2);
std::ostream& operator<<(std::ostream& os, const executable_file& f);
std::ostream& operator<<(std::ostream& os, const non_executable_file& f);
std::istream& operator>>(std::istream& is, executable_file& f);
std::istream& operator>>(std::istream& is, non_executable_file& f);
std::ostream& operator<<(std::ostream& os, const file& f);

```

dir.h

```

#pragma once
#include "file.h"

```

```

class dir { /** Клас - массив. */
private:
    file** files;
    file** copy;
    int next_ind = 0;
    int new_ind = 1;
public:
    void add_file(const executable_file& f);
    void add_file(const non_executable_file& f);
    void del_file(const int& index);
    void del_all();
    void add_file_by_str(const std::string& str);
    void read_from_file(const std::string& name);
    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&));
file* operator[](const int& index);
friend std::ostream& operator<<(std::ostream& os, dir& d);
};

```

```

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&);
std::istream& operator>>(std::istream& is, dir& d);
std::ostream& operator<<(std::ostream& os, dir& d);

```

menu.h

```

#pragma once
#include "dir.h"

```

```
void menu();
```

file.cpp

```

#include "file.h"

size_t_t file::get_x() const {
    return x;
}
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(const size_t_t& 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 по умолчанию." << "\n";
    }
    file::file(const file& f) {
        x = f.x;
        size = f.size;
        index = f.index;
        opt = f.opt;
        name = f.name;
    }
    file::file(const size_t& ver, const size_t& sz, const size_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 по умолчанию." << "\n";
    }
    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;
    }
    size_t executable_file::get_time() const {
        return runningtime;
    }
    executable_file::executable_file() : file(), runningtime(0) {
        type_of_file = 1;
    }
    executable_file::executable_file(const executable_file& sf) : file(sf),
    runningtime(sf.runningtime) {
        type_of_file = 1;
    }
    executable_file::executable_file(const size_t& x, const size_t& sz, const size_t& ind,
    const bool& sys, const bool& hid, const std::string& nm, const size_t& time) : file(x, sz,
    ind, sys, hid, nm), runningtime(time) {
        type_of_file = 1;
    }
    executable_file::~executable_file() {}
    void executable_file::change_time(const size_t& time) {
        runningtime = time;
    }
    bool non_executable_file::get_text() const {
        return is_text;
    }
    non_executable_file::non_executable_file() : file(), is_text(true) {
        type_of_file = 2;
    }
    non_executable_file::non_executable_file(const non_executable_file& nf) : file(nf),
    is_text(nf.is_text) {

```

```

        type_of_file = 2;
    }
    non_executable_file::non_executable_file(const size_t_t& x, const size_t_t& sz, const
    size_t_t& ind, const bool& sys, const bool& hid, const std::string& nm, const bool& text) :
    file(x, sz, ind, sys, hid, nm), is_text(text) {
        type_of_file = 2;
    }
    non_executable_file::~non_executable_file() {}
    void non_executable_file::change_text(const bool& text) {
        is_text = text;
    }
    std::string executable_file::get_info() const {
        std::stringstream s;
        s << runtime;
        return s.str();
    }
    std::string non_executable_file::get_info() const {
        std::stringstream s;
        s << is_text;
        return s.str();
    }
    bool operator==(const file& f1, const file& f2) {
        if (f1.get_name() != f2.get_name()) {
            return false;
        }
        else if (f1.get_sys() != f2.get_sys()) {
            return false;
        }
        else if (f1.get_size() != f2.get_size()) {
            return false;
        }
        else if (f1.get_hid() != f2.get_hid()) {
            return false;
        }
        else if (f1.get_x() != f2.get_x()) {
            return false;
        }
        else if (f1.get_index() != f2.get_index()) {
            return false;
        }
        else {
            return true;
        }
    }
    bool operator!=(const file& f1, const file& f2) {
        return !(f1 == f2);
    }
    bool operator==(const executable_file& f1, const executable_file& f2) {
        if (f1.get_name() != f2.get_name()) {
            return false;
        }
        else if (f1.get_sys() != f2.get_sys()) {
            return false;
        }
        else if (f1.get_size() != f2.get_size()) {
            return false;
        }
        else if (f1.get_hid() != f2.get_hid()) {
            return false;
        }
        else if (f1.get_x() != f2.get_x()) {
            return false;
        }
        else if (f1.get_index() != f2.get_index()) {
            return false;
        }
    }

```



```

    }
    else if (f1.get_time() != f2.get_time()) {
        return false;
    }
    else {
        return true;
    }
}
bool operator!=(const executable_file& f1, const executable_file& f2) {
    return !(f1 == f2);
}
bool operator==(const non_executable_file& f1, const non_executable_file& f2) {
    if (f1.get_name() != f2.get_name()) {
        return false;
    }
    else if (f1.get_sys() != f2.get_sys()) {
        return false;
    }
    else if (f1.get_size() != f2.get_size()) {
        return false;
    }
    else if (f1.get_hid() != f2.get_hid()) {
        return false;
    }
    else if (f1.get_x() != f2.get_x()) {
        return false;
    }
    else if (f1.get_index() != f2.get_index()) {
        return false;
    }
    else if (f1.get_text() != f2.get_text()) {
        return false;
    }
    else {
        return true;
    }
}
bool operator!=(const non_executable_file& f1, const non_executable_file& f2) {
    return !(f1 == f2);
}
bool check_str(const std::string& str){
    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;
    }
    return true;
}
std::ostream& operator<<(std::ostream& os, const file& f) {
    return os << f.type_of_file << " " << "\"" << f.get_name() << "\" " << f.get_index()
    << " " << f.get_size() << " " << f.get_x() << " " << f.get_hid() << " " << f.get_sys() << "
    " << f.get_info();
}
std::ostream& operator<<(std::ostream& os, const executable_file& f) {

```

```

        return os << f.type_of_file << " " << "\"" << f.get_name() << "\" " << f.get_index()
<< " " << f.get_size() << " " << f.get_x() << " " << f.get_hid() << " " << f.get_sys() << "
" << f.get_time();
    }
    std::ostream& operator<<(std::ostream& os, const non_executable_file& f) {
        return os << f.type_of_file << " " << "\"" << f.get_name() << "\" " << f.get_index()
<< " " << f.get_size() << " " << f.get_x() << " " << f.get_hid() << " " << f.get_sys() << "
" << f.get_text();
    }
    std::istream& operator>>(std::istream& is, executable_file& f) {
        std::string name;
        std::string temp;
        std::regex re("\\$");
        std::stringstream temps;
        executable_file tempf;
        bool check = true;
        bool global_check = true;
        do {
            is >> temp;
            if (check_str(temp)) {
                name += temp;
            }
            else {
                global_check = false;
            }
            if (std::regex_search(name, re)) {
                check = false;
            }
            else {
                name += " ";
            }
        } while (check);
        std::regex r("\\");
        name = std::regex_replace(name, r, "");
        tempf.change_name(name);
        int temp_i = 0;
        is >> temp;
        if (!check_str(temp)) {
            global_check = false;
        }
        temps << temp;
        temps >> temp_i;
        temps.clear();
        tempf.change_index(temp_i);
        is >> temp;
        if (!check_str(temp)) {
            global_check = false;
        }
        temps << temp;
        temps >> temp_i;
        temps.clear();
        tempf.change_size(temp_i);
        is >> temp;
        if (!check_str(temp)) {
            global_check = false;
        }
        temps << temp;
        temps >> temp_i;
        temps.clear();
        tempf.change_x(temp_i);
        is >> temp;
        if (!check_str(temp)) {
            global_check = false;
        }
        temps << temp;

```

```

    temps >> temp_i;
    temps.clear();
    tempf.change_hid(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_sys(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_time(temp_i);
    if (global_check == true) {
        f = tempf;
    }
    else {
        f.type_of_file = -1;
    }
    return is;
}
std::istream& operator>>(std::istream& is, non_executable_file& f) {
    std::string name;
    std::string temp;
    std::regex re("\\$");
    std::stringstream temps;
    non_executable_file tempf;
    bool check = true;
    bool global_check = true;
    do {
        is >> temp;
        if (check_str(temp)) {
            name += temp;
        }
        else {
            global_check = false;
        }
        if (std::regex_search(name, re)) {
            check = false;
        }
        else {
            name += " ";
        }
    } while (check);
    std::regex r("\\");
    name = std::regex_replace(name, r, "");
    tempf.change_name(name);
    int temp_i = 0;
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_index(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }

```

```

    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_size(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_x(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_hid(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_sys(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_text(temp_i);
    if (global_check == true) {
        f = tempf;
    }
    else {
        f.type_of_file = -1;
    }
    return is;
}
file& file::operator=(const file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    return *this;
}
executable_file& executable_file::operator=(const executable_file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    runningtime = f.runningtime;
    return *this;
}
non_executable_file& non_executable_file::operator=(const non_executable_file& f) {

```

```

        x = f.x;
        size = f.size;
        index = f.index;
        name = f.name;
        opt.change_sys(f.get_sys());
        opt.change_hid(f.get_hid());
        is_text = f.is_text;
        return *this;
    }
    std::string file::get_info() const {
        return "";
    }
    bool operator>(const file& f1, const file& f2) {
        return f1.get_name() < f2.get_name();
    }
    bool operator<(const file& f1, const file& f2) {
        return f1.get_name() > f2.get_name();
    }
    bool operator>(const executable_file& f1, const executable_file& f2) {
        return f1.get_name() < f2.get_name();
    }
    bool operator<(const executable_file& f1, const executable_file& f2) {
        return f1.get_name() > f2.get_name();
    }
    bool operator>(const non_executable_file& f1, const non_executable_file& f2) {
        return f1.get_name() < f2.get_name();
    }
    bool operator<(const non_executable_file& f1, const non_executable_file& f2) {
        return f1.get_name() > f2.get_name();
    }
}

```

dir.cpp

```

#include "dir.h"

void dir::add_file(const executable_file& f) {
    if (next_ind == 0) {
        files = new file * [next_ind + 1];
        file* ptr = new auto(f);
        files[next_ind] = ptr;
        new_ind++;
        next_ind++;
    }
    else {
        copy = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            copy[i] = files[i];
        }
        delete[] files;
        files = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            files[i] = copy[i];
        }
        file* ptr = new auto(f);
        files[next_ind] = ptr;
        delete[] copy;
        next_ind++;
        new_ind++;
    }
}

void dir::add_file(const non_executable_file& f) {
    if (next_ind == 0) {
        files = new file * [next_ind + 1];
        file* ptr = new auto(f);
        files[next_ind] = ptr;
    }
}

```

```

        new_ind++;
        next_ind++;
    }
    else {
        copy = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            copy[i] = files[i];
        }
        delete[] files;
        files = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            files[i] = copy[i];
        }
        file* ptr = new auto(f);
        files[next_ind] = ptr;
        delete[] copy;
        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];
        for (int i = 0; i < index; i++) {
            copy[i] = files[i];
        }
        for (int i = index; i < next_ind; i++) {
            copy[i] = files[i + 1];
        }
        delete files[index];
        delete[] files;
        files = copy;
        copy = 0;
    }
}

void dir::del_all() {
    if (next_ind != 0) {
        for (int i = 0; i < next_ind; i++) {
            delete files[i];
        }
        delete[] files;
        next_ind = 0;
    }
}

void dir::read_from_file(const std::string& name) {
    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);
        delete[] temp;
    }
    f.close();
}

std::string dir::get_file_to_str(const int& index) const {
    std::stringstream s;
    s << files[index];
    return s.str();
}

```

```

}
void dir::print_all() const {
    for (int i = 0; i < next_ind; i++) {
        std::cout << i + 1 << " " << (*(files+i)) << "\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) {
    if (check_str(str)) {
        std::stringstream s;
        std::regex re("\\.+"");
        std::smatch sm;
        std::string temp;
        std::regex_search(str, sm, re);
        auto i = str.find("\\");
        i = str.find("\\", i + 1);
        std::regex re_temp("\\");
        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);
        s << temp;
        auto index = 0;
        int check = 0;
        s >> check;
        s.clear();
        i = i2;
        i2 = str.find(" ", i + 1);
        temp = str.substr(i + 1, i2 - i);
        s << temp;
        s >> index;
        s.clear();
        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;
        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;
        auto i7 = str.find(" ", i6 + 1);
        temp = str.substr(i6 + 1, i7 - i6);
        s.clear();
        s << temp;
        if (check == 1) {
            size_t_t time;
            s >> time;
            executable_file new_file(x, size, index, sys, hid, name, time);
            add_file(new_file);
        }
        else {
            bool text;
            s >> text;
            non_executable_file nex_file(x, size, index, sys, hid, name, text);
            add_file(nex_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 re1("[A-Za-zA-Яa-я0-9\\s\\!,\\?\\\"\\.\\;\\'"]*");
    if (!(std::regex_search(str, re1))) {
        return false;
    }
    std::regex re2("\\s{2,}");
    if (std::regex_search(str, re2)) {
        return false;
    }
    std::regex re3("[\\!\\?\\.\\;\\,]{2,}");
    if (std::regex_search(str, re3)) {
        return false;
    }
    std::regex re4("[\\'\\\"]{2,}");
    if (std::regex_search(str, re4)) {
        return false;
    }
    std::regex re5("^\\s[A-ZA-Я]");
    if (!std::regex_search(str, re5)) {
        return false;
    }
    return true;
}

void dir::print_all_with_2_or_more_words() const {
    for (int i = 0; i < next_ind; i++) {
        std::regex re1("\\.\\.\\.");
        std::string str = files[i]->get_name();
        if (std::regex_search(str, re1)) {

```



```

        std::cout << i + 1 << " " << (*(files + i)) << "\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);
}

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());
}

std::istream& operator>>(std::istream& is, dir& d) {
    int temp;
    executable_file exetemp;
    non_executable_file nontemp;
    while (is >> temp) {
        if (temp == 1) {
            is >> exetemp;
            if (exetemp.type_of_file != -1) {
                d.add_file(exetemp);
            }
        }
        else {
            is >> nontemp;
            if (nontemp.type_of_file != -1) {
                d.add_file(nontemp);
            }
        }
    }
}

```

```

    }
    return is;
}
std::ostream& operator<<(std::ostream& os, dir& d) {
    for (size_t i = 0; i < d.next_ind; i++) {
        os << *(d[i]) << "\n";
    }
    return os;
}
file* dir::operator[](const int& index) {
    return files[index];
}

```

menu.cpp

```

#include "menu.h"

void menu() {
    setlocale(LC_ALL, "Russian"); /** Локалізація консолі. */
    int n = 0, temp_i;
    dir directory;
    std::ifstream f("data.txt");
    std::ofstream d;
    f >> directory;
    f.close();
    int s;
    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");
            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) {
            std::cout << directory;
        }
        else if (n == 5) {
            std::cout << "Количество скрытых системных файлов: " << directory.count_system()
<< "\n";
        }
        else if (n == 6) {
            f.open("data.txt");
            f >> directory;
            f.close();
        }
        else if (n == 7) {
            d.open("data.txt");
            d << directory;
            d.close();
        }
    }
}

```

```

        else if (n == 8) {
            directory.print_all_with_2_or_more_words();
        }
        else if (n == 9) {
            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) {
                directory.sort_files(sort_version);
            }
            else if (s == 2) {
                directory.sort_files(sort_size);
            }
            else if (s == 3) {
                directory.sort_files(sort_ind);
            }
            else if (s == 4) {
                s = 0;
                directory.sort_files(sort_opt);
            }
            else if (s == 5) {
                directory.sort_files(sort_name);
            }
        }
    }
    directory.del_all();
}

```

ПГ3.cpp

```

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

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

```

tests.cpp

```

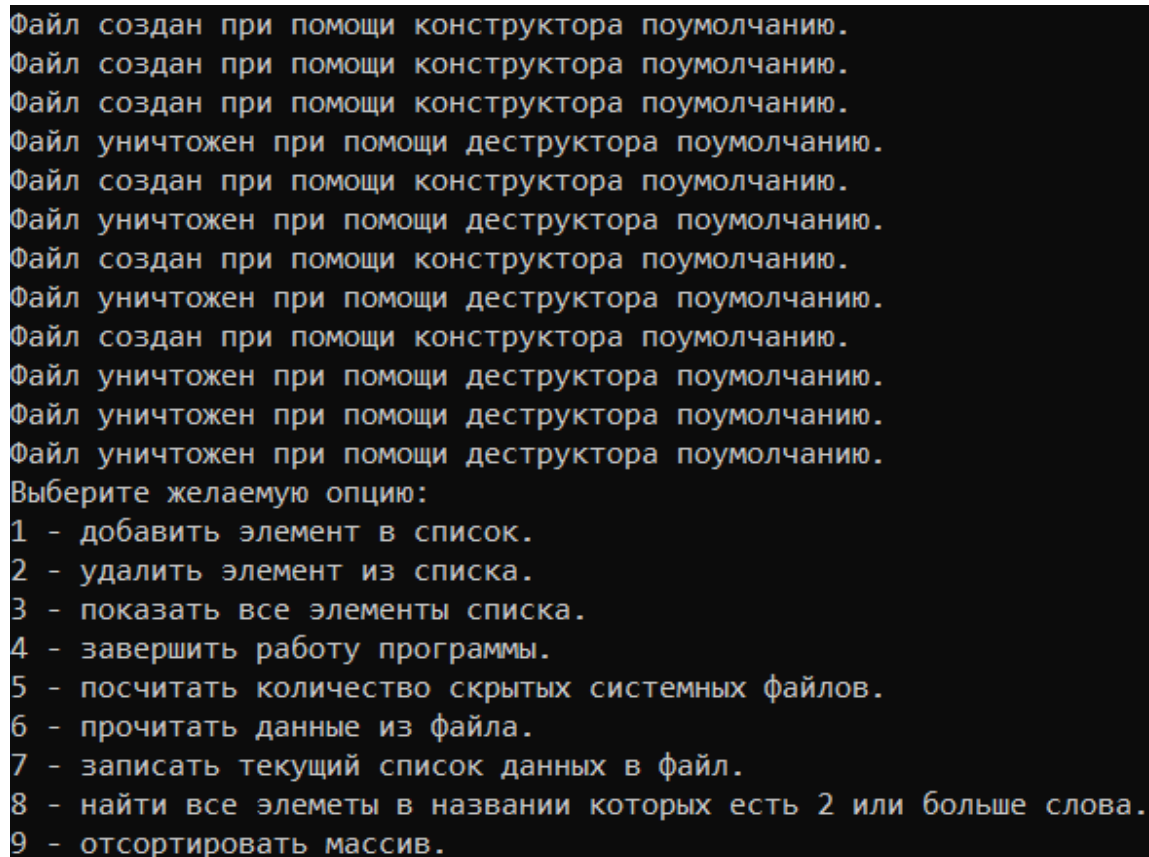
#include "menu.h"
#define _CRTDBG_MAP_ALLOC

int main() {
    setlocale(LC_ALL, "Russian");
    std::ifstream in("test1.txt");
    std::ofstream on("test2.txt");
    int n;
    dir d;
    in >> d;
    on << d;
    std::cout << "Если содержимое файлов test1 и test2 теперь совпадают, то тест пройден.
Нажмите любую клавишу для завершения работы программы. ";
    if (_CrtDumpMemoryLeaks()) {
        std::cout << "Утечка памяти обнаружена." << "\n";
    }
    else {
        std::cout << "Утечка памяти не обнаружена." << "\n";
    }
    std::cin >> n;
}

```

Додаток Б

Результати роботи програми



Файл создан при помощи конструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
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 - прочитать 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.

Рисунок 1 – Результаты работы програми