Звіт

Автор: Богданов І.Ю. КІТ-119а Дата: 17 квітня 2020

Лабораторна робота №6. Спадкування

Тема. Класи. Спадкування.

Мета: отримати знання про парадигму ООП – спадкування. Навчитися

застосовувати отримані знання на практиці.

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

Змінити попередню лабораторну роботу й додати клас нащадок базового класу.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: file

Клас нащадок базового класу: system_file

Клас, що має в собі масив базового класу та методи для роботи з ним: dir й його аналог для класу нащадку sys_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 программа).

`size_t_t size` – поле класу file та його нащадка(розмір файлу у бітах).

`size_t_t index` – поле класу file та його нащадка(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 та його нащадка(назва файлу).

`x_version* x` – поле класу file та його нащадка(64 або 32 розрядна

программа).

`options opt` поле класу file та його нащадка(чи є файл прихованим і системним).

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

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

`file* files` – поле класу dir та його аналога, в якому пристосовано для роботи з класом system_file (масив елементів класу file).

`file* copy` – поле класу dir та його аналога, в якому пристосовано для роботи з класом system_file(показчик на клас file, використовується для правильної роботи деяких методів).

`bool is_executable` – поле класу system_file(чи є даний файл виконуваним).

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).

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

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

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

`virtual bool get_sys() const` – отримання значення атрибуту is_system змінної класу file(метод класів file й system_file).

`virtual bool get_hid() const` – отримання значення атрибуту is_hidden змінної класу file(метод класів file й system_file).

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

`virtual void change_x(x_version*)` – зміна значення поля x змінної класу file(метод класів file й system_file).

`virtual void change_size(const size_t_t &sz)` – зміна значення поля size змінної класу file(метод класів file й system_file).

`virtual void change_index(const size_t_t &in)` – зміна значення поля index змінної класу file(метод класів file й system_file).

`virtual void change_sys(const bool&)` – зміна значення атрибуту is_system змінної класу file(метод класів file й system_file).

`virtual void change_hid(const bool&)` – зміна значення атрибуту is_hidden змінної класу file(метод класів file й system_file).

`virtual void change_name(const std::string&)` – зміна значення поля name змінної класу file(метод класів file й system_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 класів dir й sys_dir).

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

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

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

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

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

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

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

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

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

`void sort_files(bool (*f)(file&,file&))` – сортування усіх об'єктів класу file в об'єкті класу dir на екран(метод класів dir й sys_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).
`virtual bool get_exe() const` – метод класу file перевантажений у класі system_file.
`virtual bool get_exe() const override final` – отримання значення поля is_executable змінної класу system_file(метод класу system_file).
`virtual void change_exe(const bool&) final` – зміна значення поля is_executable змінної класу system_file(метод класу system_file).
`system_file()` – конструктор класу system_file.
`system_file(const system_file&)` – конструктор копіювання класу system_file.
`system_file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const std::string&, const bool&)` – конструктор з параметрами класу system_file.
`~system_file()` – деструктор класу system_file.

2.4 Опис функцій

`void menu()` – функція меню.
`bool sort_version(file&, file&)` – функція порівняння двух файлів по x, має перевантаження для роботи з класом system_file.
`bool sort_size(file&, file&)` – функція порівняння двух файлів по розміру, має перевантаження для роботи з класом system_file.
`bool sort_ind(file&, file&)` – функція порівняння двух файлів по індексу, має перевантаження для роботи з класом system_file.
`bool sort_opt(file&, file&)`– функція порівняння двух файлів по тому чи є файл прихованим або системним, має перевантаження для роботи з класом system_file.
`bool sort_name(file&, file&)`– функція порівняння двух файлів по назві, має перевантаження для роботи з класом system_file.
`bool sort_exe(system_file&, system_file&)` функція порівняння двух системних файлів по тому чи можна їх запустити.

3 Текст програми

Лабораторная работа 6.cpp

```
#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>

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:
	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(x_version*);
	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 bool get_exe() const;
	file();
	file(const file&);
	file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const
std::string&);
	~file();
};
class system_file final : public file {
private:
	bool is_executable;
public:
	virtual bool get_exe() const override final;
	virtual void change_exe(const bool&) final;
	system_file();
	system_file(const system_file&);
	system_file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const
std::string&, const bool&);
	~system_file();
};
```
file.cpp

```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;
}
bool file::get_exe() const {
	return false;
}
bool system_file::get_exe() const {
	return is_executable;
}
system_file::system_file(): file(), is_executable(false){ }
system_file::system_file(const system_file& sf) : file(sf), is_executable(sf.is_executable){ }
system_file::system_file(x_version* x, const size_t_t& sz, const size_t_t& ind, const bool& sys,
const bool& hid, const std::string& nm , const bool& exe) : file(x, sz, ind, sys, hid, nm),
is_executable(exe) { }
system_file::~system_file() {}
void system_file::change_exe(const bool& exe) {
	is_executable = exe;
}
```

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*);
	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&);

class sys_dir {
private:
	system_file* files;
	system_file* copy;
	int next_ind = 0;
	int new_ind = 1;
```

```
public:
	void add_file(const system_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*);
	std::string get_file_to_str(const int& index) const;
	system_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)(system_file&, system_file&));
};

bool sort_version(system_file&, system_file&);
bool sort_size(system_file&, system_file&);
bool sort_ind(system_file&, system_file&);
bool sort_opt(system_file&, system_file&);
bool sort_name(system_file&, system_file&);
bool sort_exe(system_file&, system_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 (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];
		}
		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-Яа-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-ZA-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);
}
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());
}

void sys_dir::add_file(const system_file& f) {
	if (next_ind == 0) {
		files = new system_file[1];
		files[next_ind] = f;
		files[next_ind].change_index(new_ind);
		new_ind++;
		next_ind++;
	}
	else {
		copy = new system_file[next_ind];
		for (int i = 0; i < next_ind; i++) {
			copy[i] = files[i];
```

```
			}
			delete[] files;
			files = new system_file[next_ind + 1];
			for (int 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 sys_dir::del_file(const int& index) {
		if (next_ind == 1) {
			delete[] files;
			next_ind--;
		}
		else {
			copy = new system_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 system_file[next_ind - 1];
			for (int i = 0; i < next_ind - 1; i++) {
				files[i] = copy[i];
			}
			delete[] copy;
			next_ind--;
		}
	}
	void sys_dir::del_all() {
		if (next_ind != 0) {
			delete[] files;
			next_ind = 0;
		}
	}
	void sys_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 sys_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() << " " << files[index].get_exe();
		return s.str();
	}
	void sys_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 sys_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;
}
system_file sys_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 sys_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;
		auto i7 = str.find(" ", i6 + 1);
		s.clear();
		bool exe = false;
		temp = str.substr(i6 + 1, i7 - i6);
		s << temp;
		s >> exe;
```

```
			system_file new_file(temp_x, size, index, sys, hid, name, exe);
			add_file(new_file);
		}
	}
	void sys_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 sys_dir::check_str(const std::string& str) const {
		std::regex re("[A-Za-zA-Яа-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-ZA-Я]");
		if (!std::regex_search(str, re_5)) {
			return false;
		}
		return true;
	}
	void sys_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 sys_dir::sort_files(bool (*f)(system_file&, system_file&)) {
		bool check = false;
		system_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(system_file& first, system_file& second) {
		return (first.get_x() < second.get_x());
	}
	bool sort_size(system_file& first, system_file& second) {
```

```
	return (first.get_size() < second.get_size());
}
bool sort_ind(system_file& first, system_file& second) {
	return (first.get_index() > second.get_index());
}
bool sort_opt(system_file& first, system_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(system_file& first, system_file& second) {
	return (first.get_name() > second.get_name());
}
bool sort_exe(system_file& first, system_file& second) {
	return (first.get_exe() < second.get_exe());
}
```

menu.h

```
#pragma once
#include "dir.h"

void menu();
```

menu.cpp

```
#include "menu.h"

void menu() {
    setlocale(LC_ALL, "Russian");
    std::cout << "Работа с базовым классом - 1, с классом наследником - 2: ";
    int n = 0, temp_i;
    std::cin >> n;
    x_version* x32 = new x_version;
    x32->change_ver(32);
    x_version* x64 = new x_version;
    x64->change_ver(64);
    if (n == 1) {
        dir directory;
        directory.read_from_file("data.txt", x32, x64);
        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", x32, x64);
                std::cout << "Файл добавлен." << "\n";
            }
            else if (n == 2) {
                std::cout << "Введите номер удалемого элемента (нумерация начинаеться 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 << "Ввведите 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();
    }
    if (n == 2) {
        sys_dir directory;
        directory.read_from_file("data2.txt", x32, x64);
        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", x32, x64);
                std::cout << "Файл добавлен." << "\n";
            }
            else if (n == 2) {
                std::cout << "Введите номер удалемого элемента (нумерация начинаеться 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("data2.txt", x32, x64);
            }
            else if (n == 7) {
                directory.print_to_file("data2.txt");
            }
            else if (n == 8) {
                directory.print_all_with_2_or_more_words();
            }
            else if (n == 9) {
                std::cout << "Ввведите номер признака по которому хотите отсортировать массив: 1 -
x, 2 - size, 3 - index, 4 - opt, 5 - name, 6 - exe. " << "\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);
                }
                else if (s == 6)
                    directory.sort_files(sort_exe);
            }
        }
        directory.del_all();
    }
    delete x32;
    delete x64;
}
```

```
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
data2.txt
"Sys file" 2 123 32 1 1 1
"Qwerty" 3 521 64 1 0 0
"Hello friend" 4 289 64 0 1 0
"Water" 5 10000 32 0 0 1
```

4. Результати роботи програми

Результати роботи програми:

```
Работа с базовым классом - 1, с классом наследником - 2: 2
Файл создан при помощи конструктора с аргументами.
Файл создан при помощи конструктора поумолчанию.
Файл уничтожен при помощи деструктора поумолчанию.
Файл создан при помощи конструктора с аргументами.
Файл создан при помощи конструктора поумолчанию.
Файл уничтожен при помощи деструктора 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 создан при помощи конструктора с аргументами.
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 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 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 - прочитать данные из файла.
7 - записать текущий список данных в файл.
8 - найти все элеметы 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 1 0 0 0 1
```

5. Висновки

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи наслідуванням.

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