МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ХПІ»

Кафедра «Обчислювальна техніка та програмування»

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.119а.

Розробники
Виконав:
студент групи КІТ-119а
_____/ Татаренко А.Г./

Перевірив:
_____/аспірант Бартош М. В./

Харків 2020

ЗАТВЕРДЖЕНО

КІТ.119а.

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.119а.

Аркушів _34_

Харків 2020

ЗМІСТ

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 академічна група, а саме про студентів. 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 елементу, 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 елементу, 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 непреривної області для зберігання.

До переваг списку можна віднести:

1. Списки досить ефективні щодо операцій додавання або видалення елементу 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 *Student* та класи-спадкоємці *Course* та *Surnames*, які спадкують поля базового класу.

На рис. 1 показано внутрішню структуру, а на рис. 2 - відносини розроблених класів у вигляді UML-діаграми.

а) б) в)

Рис. 1. Поля базового класу та класів-спадкоємців (рис. 1а-в)

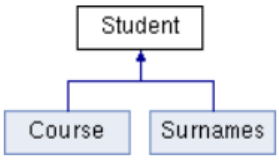


Рис. 2. Схема ієрархії розроблених класів

Дані про програми будуть заноситися до списку. Для цього було розроблено клас-контролер *CList* з полями, показаними на рис. 3, і методами на рис. 4.

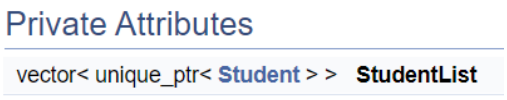


Рис. 3. Поля класу-контролеру *CList*

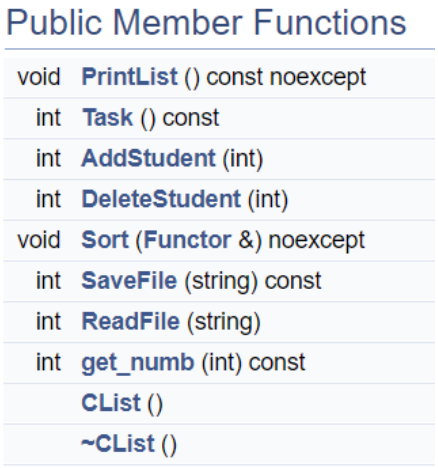
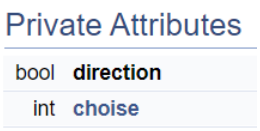


Рис. 4. Розроблені методи класу *CList*

Так як метод сортування було реалізовано за допомогою функтора, його поля та методи можна побачити на рис. 5а-б.



а)

б)

Рис. 5. Поля та методи класу-функтора

Також відповідно до додаткового завдання було створено клас-тестер, який слугує виконує функції тестування основних методів класу-контролера. Його поля та методи можна побачити на рис. 6а-б.

а) б)

Рис. 6. Поля та методи класу-тестеру

На рис. 7 подано структуру проекту розробленого програмного продукту.

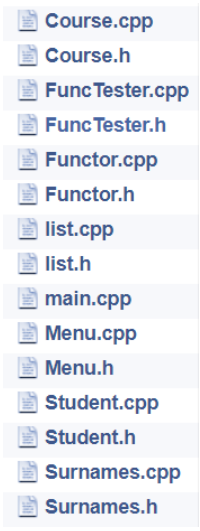


Рис. 7. Структура проекту

СХЕМИ АЛГОРИТМУ ПРОГРАМИ

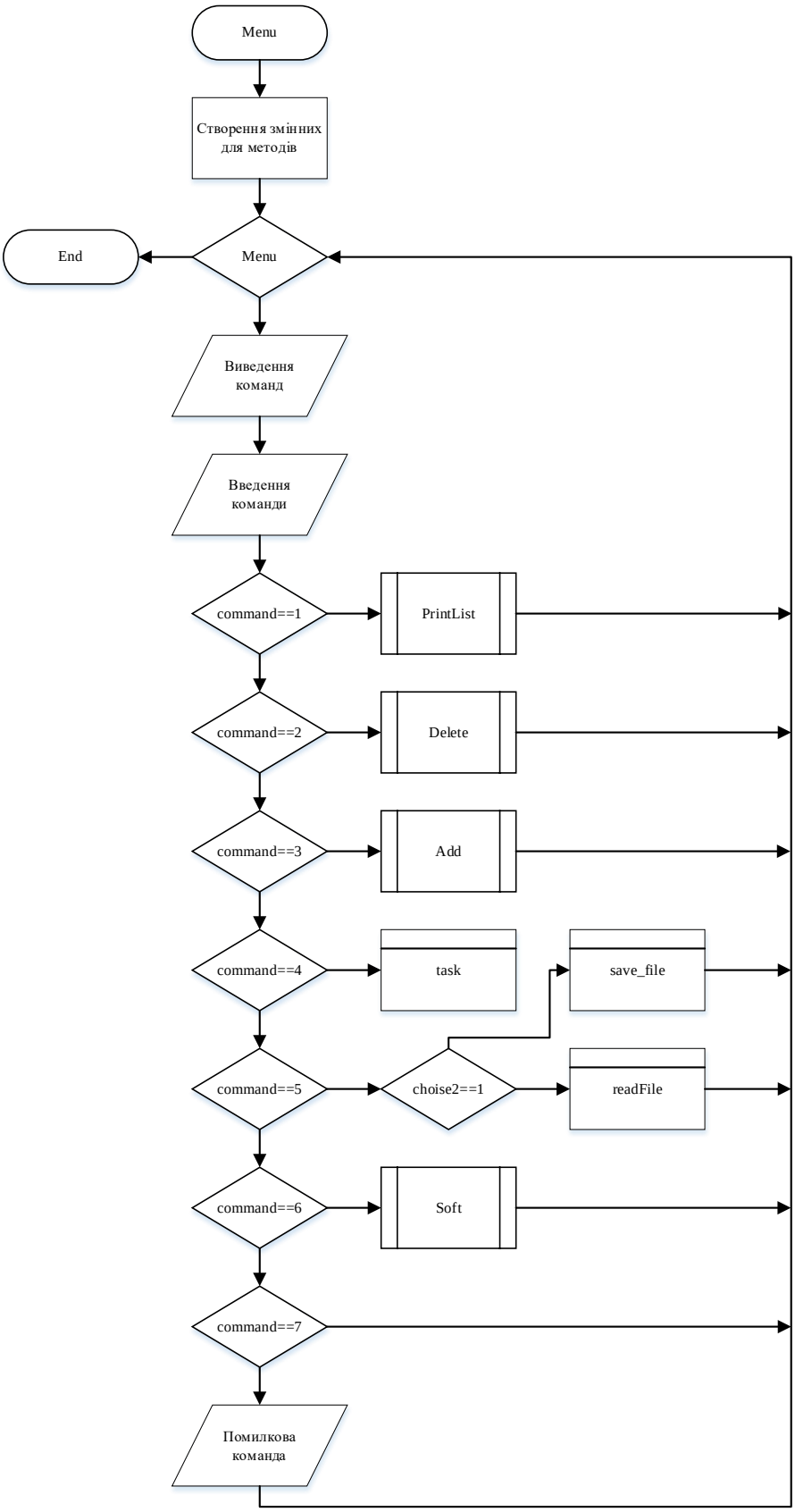


Рис. 8. Схема алгоритму методу Menu

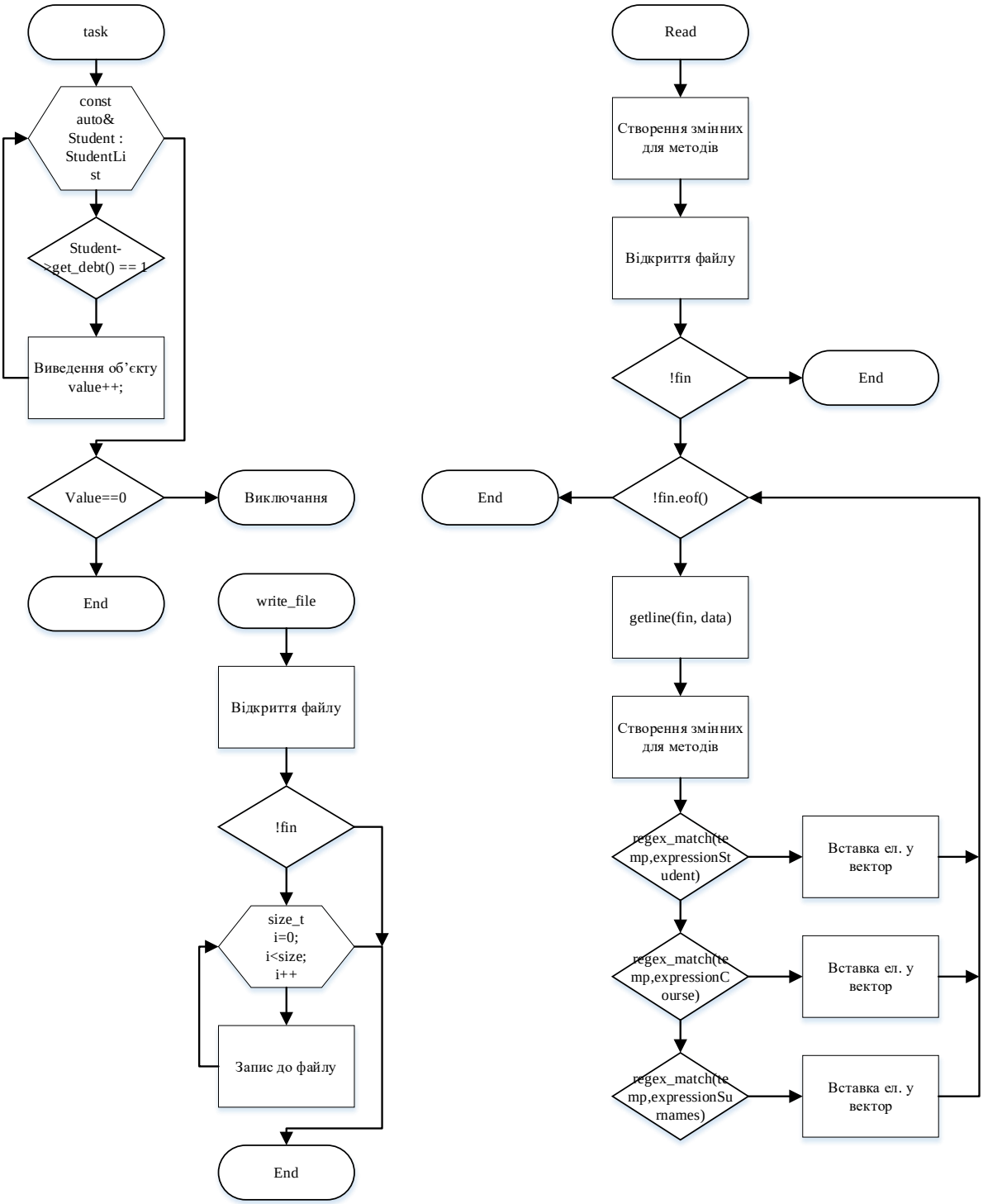


Рис. 9. Схеми алгоритмів методів ReadFile, WriteFile, Task

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 література 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 колекцію розроблених класів, та наступні методи роботи з цією колекцією:

   а) читання даних з файлу та їх запис у

   контейнер; б) запис даних з контейнера у файл;

   в) сортування елементів у контейнері за вказаними критеріями:

   поле та напрям сортування, які задаються користувачем з клавіатури;

   г) Виконання індивідуального завдання;

   д) Додавання елементів у контейнер;

   е) Видалення елементів з контейнеру;
4. Розроблено клас, який відображає діалогове меню для демонстрації реалізованих функцій класу контролера;
5. Оформлено схеми алгоритмів функцій класів контролера та діалогового меню;
6. Оформлено документацію;
7. Було додано обробку помилок, перевірку вхідних даних за допомогою регулярних виразів;
8. Розроблено клас-тестер, що перевіряє методи класу-контролера на коректність.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Шілдт, Герберт. Повний довідник по C++, 4-е видання .: Пер. з англ .: - М .: Изд. будинок «Вільямс», 2004. - 800 с .;
2. Дейтел Х.М. Як програмувати на C++ / Х.М. Дейтел, П.Дж. Дейтел М.: ЗАТ БІНОМ, 1999. - 1000 с.
3. Штейн Кліфорд (2019). Алгоритми. Побудова і аналіз.
4. Вандервуд, Джосаттіс - Шаблони C++. Довідник розробника. / Пер. з англ. - М .: Вільямс, 2008. - 536 с.
5. Андрій Александреску, Сучасне проектування на C++. М.: ТОВ «І.Д.Вільямс», 2002.
6. Страуструп Б. Дизайн і еволюція C++ / Б. Страуструп; пер. з англ. - М. : ДМК Прес; С.Пб: Пітер, 2007. - 445 с.
7. Остерн. Узагальнене програмування і STL: Використання інаращіваніе стандартної бібліотеки шаблонів C++ / Остерн; Пер. Санглена. - С.Пб: Невський Діалект, 2004. - 544 с.

Додаток А

Текст програми

Course.h

```
/**
* @file Course.h
* Файл оголошення класу спадкоємця
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#pragma once
#include "Student.h" /** Підключення файлу Student.h */

class Course final : public Student /** Оголошення класу спадкоємця */
{
private:
	int course; /** Підключення файлу Student.h */

public:
	int get_age() const override; /** Оголошення перевантаженого гетера віку студента */
	int get_prog_d() const override; /** Оголошення перевантаженого гетера боргу з програмування */
	int get_mark() const override; /** Оголошення перевантаженого гетера середньої оцінки */
	int get_number()const override; /** Оголошення перевантаженого гетера номера студента */
	bool get_debt()const override; /** Оголошення перевантаженого гетера боргу студента */
	string get_name() const override; /** Оголошення перевантаженого гетера імені студента */

	string getInfo() const override; /** Оголошення перевантаженого метода отримання інформації студента */
	void enter(istream&) override; /** Оголошення перевантаженого метода введення інформації студента */

	Course(); /** Оголошення конструктора за замовчуванням */
	Course(int, int, int, string, bool, int, int); /** Оголошення конструктора з параметрами */
	Course(const Course&); /** Оголошення конструктора копіювання */
	~Course() override; /** Оголошення перевантаженного деструктора */

	Course& operator= (Course&); /** Оголошення перевантаженого оператора присвоювання */
	bool operator!=(const string) const override; /** Оголошення перевантаженого оператора нерівності */
	bool operator==(const int) const override; /** Оголошення перевантаженого оператора порівняння */
};
```

FuncTester.h

```
/**
* @file FuncTester.h
* Файл оголошення класу-тестера
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#pragma once
#include "list.h" /** Підключення файлу list.h */

class FuncTester /** Оголошення класу-тестера */
```

```
{
private:
	CList value; /** Змінна класу-контролера */

public:
	void Sort_Test(); /** Оголошення метода тестування сортування */
	void Add_Test(); /** Оголошення метода тестування додавання елементів */
	void Delete_Test(); /** Оголошення метода тестування видалення елементів */
	void Task_Test(); /** Оголошення метода тестування виконання індивідуального завдання
*/
	void ReadFile_Test(); /** Оголошення метода тестування читання з файлу */
	void SaveFile_Test(); /** Оголошення метода тестування запису даних у файл */

	FuncTester(); /** Оголошення конструктора за замовчуванням */
	~FuncTester(); /** Оголошення деструктора */
                                                };
```

Functor.h

```
/**
* @file Functor.h
* Файл оголошення класу, який виконує функції функтора
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#pragma once
#include "Student.h" /** Підключення файлу Student.h */

class Functor /** Оголошення класу функтора*/
{
private:
	bool direction; /** Напрям сортування*/
	int choise; /** Вибір поля сортування*/

public:
	bool operator()(const unique_ptr<Student>& first, const unique_ptr<Student>& second)
const; /** Оголошення перевантаженого оператора () */

	Functor(bool, int); /** Оголошення конструктора по замовчуванням*/
	~Functor(); /** Оголошення деструктора*/
                                                };
```

list.h

```
/**
* @file list.h
* Файл оголошення класу-контролера.
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#pragma once
#include "Surnames.h" /** Підключення файлу Surnames.h */
#include "Course.h" /** Підключення файлу Course.h */
#include "Functor.h" /** Підключення файлу Functor.h */

class CList /** Оголошення класу-контролера */
{
private:
	vector <unique_ptr<Student>> StudentList; /** Контейнер для зберігання елементів */

public:
	void PrintList() const noexcept; /** Оголошення методу виводу списку у консоль */
	int Task() const; /** Оголошення методу виконання індивідуального завднання */
	int AddStudent(int); /** Оголошення методу додавання нового елемента у список */
	int DeleteStudent(int); /** Оголошення методу видалення програми зі списку */
```

```
        void Sort(Functor&) noexcept; /** Оголошення методу сортування даних */
        int SaveFile(string) const; /** Оголошення методу виведення даних у файл */
        int ReadFile(string); /** Оголошення методу виводу списку у консоль */
        int get_numb(int) const; /** Оголошення методу отримання індекса за номером */

        CList(); /** Оголошення конструктора за замовчуванням */
        ~CList(); /** Оголошення деструктора */
                                                };
```

Menu.h

```
/**
* @file Menu.h
* Файл оголошення класу меню
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#pragma once
#include "list.h" /** Підключення файлу list.h */

class Menu
{
public:
        void menu() const; /** Оголошення методу роботи зі списком*/

        Menu(); /** Оголошення конструктора за замовчуванням */
        ~Menu(); /** Оголошення деструктором */
                                                };
```

Student.h

```
/**
* @file Student.h
* Підключення необхідних бібліотек та оголошення класу Student.
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#pragma once

#define _CRT_SECURE_NO_WARNINGS
#define CRTDBG_MAP_ALLOC
#include <crtdbg.h> /** Підключення бібліотеки crtdbg.h*/
#define DEBUG_NEW new(_NORMAL_BLOCK, FILE, __LINE)

#include <string> /** Підключення бібліотеки string*/
#include <iostream> /** Підключення бібліотеки iostream*/
#include <iomanip> /** Підключення бібліотеки iomanip*/
#include <locale> /** Підключення бібліотеки locale*/
#include <fstream> /** Підключення бібліотеки fstream*/
#include <sstream> /** Підключення бібліотеки sstream*/
#include <regex> /** Підключення бібліотеки regex*/
#include <memory> /** Підключення бібліотеки memory*/
#include <vector> /** Підключення бібліотеки vector*/
#include <exception> /** Підключення бібліотеки exception*/
#include <iterator> /** Підключення бібліотеки iterator*/

using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::setw;
using std::boolalpha;
using std::setiosflags;
using std::ios;
using std::ifstream;
```

```
using std::istream;
using std::ostream;
using std::ofstream;
using std::stringstream;
using std::istringstream;
using std::regex;
using std::regex_match;
using std::regex_search;
using std::regex_replace;
using std::cmatch;
using std::unique_ptr;
using std::vector;
using std::exception;
using std::iterator;

class Student { /** Оголошення базового класу*/
protected:
	int age; /** Вік студента*/
	int number_stud; /** Номер студента*/
	int middle_mark; /** Середній бал*/
	string name; /** Ім'я*/
	bool debt; /** Чи є у студента борг*/
	int prog_d; /** Борг з програмування*/

public:
	virtual int get_age() const; /** Оголошення віртуального гетера отримання часу роботи програми*/
	virtual int get_prog_d() const; /** Оголошення віртуального гетера отримання боргу з програмування*/
	virtual int get_mark() const; /** Оголошення віртуального гетера отримання середньої оцінки*/
	virtual int get_number()const; /** Оголошення віртуального гетера отримання номера студента*/
	virtual bool get_debt()const; /** Оголошення віртуального гетера отримання часу роботи програми*/
	virtual string get_name() const; /** Оголошення віртуального гетера отримання імені студента*/

	virtual string getInfo() const; /** Оголошення віртуальної функції отримання інформації студента*/
	virtual void enter(istream&);

	Student(); /** Оголошення конструктора по замовчуванням*/
	Student(int, int, int, string, bool, int); /** Оголошення конструктора з параметрами*/
	Student(const Student& other); /** Оголошення конструктора копіювання*/
	virtual ~Student(); /** Оголошення віртуального деструктора*/

	friend ofstream& operator<< (ofstream&, const Student&); /** Оголошення перевантаженого оператора виводу у файл*/
	friend ostream& operator<< (ostream&, const Student&); /** Оголошення перевантаженого оператора виводу у консоль*/
	friend istream& operator>> (istream&, Student&); /** Оголошення перевантаженого оператора вводу*/
	virtual bool operator!=(const string) const; /** Оголошення віртуального перевантаженого оператора нерівності*/
	virtual bool operator==(const int) const; /** Оголошення віртуального перевантаженого оператора порівняння*/
	Student& operator= (Student&); /** Оголошення перевантаженого оператора присвоювання*/
                                        };
```

Surnames.h

```
/**
* @file Surnames.h
* Файл оголошення класу-спадкоемця.
```

```
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#pragma once
#include "Student.h" /** Підключення файлу Student.h */

class Surnames final : public Student /** Оголошення класу спадкоємця */
{
private:
	string star; /** Прізвище старости */

public:
	int get_age() const override; /** Оголошення перевантаженого гетера отримання віку студента */
	int get_prog_d() const override; /** Оголошення перевантаженого гетера отримання номера студента */
	int get_mark() const override; /** Оголошення перевантаженого гетера отримання середньої оцінки */
	int get_number()const override; /** Оголошення перевантаженого гетера отримання боргу з програмування */
	bool get_debt()const override; /** Оголошення перевантаженого гетера отримання віку студента */
	string get_name() const override; /** Оголошення перевантаженого гетера отримання імені студента */

	string getInfo() const override; /** Оголошення перевантаженого метода отримання інформації студента */
	void enter(istream&) override; /** Оголошення перевантаженого метода вводу інформації студента */

	Surnames(); /** Оголошення конструктора за замовчуванням */
	Surnames(int, int, int, string, bool, int, string); /** Оголошення конструктора з параметрами */
	Surnames(const Surnames&); /** Оголошення конструктора копіювання */
	~Surnames() override; /** Оголошення перевантаженого деструктора */

	Surnames& operator= (Surnames&); /** Оголошення оператора присвоювання */
	bool operator!=(const string) const override; /** Оголошення перевантаженого оператора нерівності */
	bool operator==(const int) const override; /** Оголошення перевантаженого оператора порівняння */
};
```

Course.cpp

```
/**
* @file Course.cpp
* Файл реалізації методів класу-спадкоємця
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#include "Course.h" /** Підключення файлу Course.h */

int Course::get_age() const /** Реалізація геттера віку студента */
{
	return age; /** Повернення віку студента */
}

int Course::get_prog_d() const /** Реалізація геттера боргу з програмування */
{
	return prog_d; /** Повернення боргу з програмування */
}
```

```
int Course::get_mark() const /** Реалізація геттера середньої оцінки */
{
	return middle_mark; /** Повернення середньої оцінки */
}

int Course::get_number() const /** Реалізація геттера номера студента */
{
	return number_stud; /** Повернення номера студента */
}

bool Course::get_debt()const /** Реалізація геттера боргу студента */
{
	return debt; /** Повернення змінної боргу */
}

string Course::get_name()const /** Реалізація геттера імені студента */
{
	return name; /** Повернення імені студента */
}

string Course::getInfo() const /** Реалізація функції отримання інформації */
{
	stringstream temp; /** Оголошення змінної stringstream */
	temp.setf(ios::left);

	temp << setw(10) << age << setw(8) << number_stud << setw(16) << middle_mark <<
setw(9)
		<< name << setw(7) << debt << setw(14) << prog_d << setw(4) << course; /**
Отримання рядка з даними */

	return temp.str(); /** Повернення рядка з інформацією */
}

void Course::enter(istream& data) /** Реалізація перевантаженого оператора вводу */
{
	data >> age >> number_stud >> middle_mark >> name >> debt >> prog_d >> course; /**
Введення даних у об'єкт типу istream */
}

Course::Course(int a, int n, int m, string na, bool d, int pd, int c) : Student(a, n, m, na,
d, pd), course(c) {} /** Реалізація конструктора з параметрами */
Course::Course() : Student(), course(1) {} /** Реалізація конструктора за замовчуванням */
Course::Course(const Course& other) : Student(other), course(other.course) {} /** Реалізація
конструктора копіювання */
Course::~Course() {} /** Реалізація деструктора */

Course& Course::operator=(Course& temp) /** Реалізація перевантаження оператора присвоювання
*/
{
	if (this == &temp) /** Перевірка якщо змінні однакові */
		return *this;

	Student::operator=(temp); /** Присвоювання полів базового класу */
	int course = temp.course; /** Присвоювання полів класу-спадкоємця */

	return *this; /** Повернення покажчика */
}

bool Course::operator!=(const string type) const /** Реалізація перевантаження оператора
нерівності */
{
	if (this->course != 1) /** Перевірка відбувається по типу зловмисного ПО */
		return true; /** Якщо програма не є трояном */
	else
		return false; /** Якщо програма є трояном */
}
```

```
bool Course::operator==(const int number_stud) const /** Реалізація перевантаження оператора порівняння */
{
	return this->number_stud == number_stud; /** Перевірка відбувається по номеру */
}
```

FuncTester.cpp

```
/**
* @file FuncTester.cpp
* Файл реалізації методів класу-тестера
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#include "FuncTester.h"                                    /** Підключення файлу FuncTester.h */

void FuncTester::Add_Test()                                /** Реалізація тестування метода додавання студента */
{
	if (value.AddStudent(2) == 11)                     /** Виклик метода додавання студента */
		cout << "Тест функции добавления студента\t выполнен успешно." << endl;
	else
		cout << "Тест функции добавления студента\t не выполнен успешно." << endl;
}

void FuncTester::Delete_Test()                             /** Реалізація тестування метода видалення студента */
{
	if (value.DeleteStudent(5) == 10)                  /** Виклик метода видалення студента */
		cout << "Тест функции удаления студента\t\t выполнен успешно." << endl;
	else
		cout << "Тест функции удаления студента\t\t не выполнен успешно." << endl;
}

void FuncTester::ReadFile_Test()                           /** Реалізація тестування метода читання даних з файлу */
{
	if (value.ReadFile("data.txt") == 5)               /** Виклик метода читання даних з файлу */
		cout << "Тест функции чтения файла\t\t выполнен успешно." << endl;
	else
		cout << "Тест функции чтения из файла\t\t не выполнен успешно." << endl;
}

void FuncTester::SaveFile_Test()                           /** Реалізація тестування метода виведення даних у файл */
{
	if (value.SaveFile("Test.txt") == 10)              /** Виклик метода виведення даних у файл */
		cout << "Тест функции сохранения в файл\t\t выполнен успешно." << endl;
	else
		cout << "Тест функции сохранения в файл\t\t не выполнен успешно." << endl;
}

void FuncTester::Sort_Test()                               /** Реалізація тестування метода сортування */
{
	Functor funct(1, 2);                               /** Створення елемента класу Functor */
	int beforeSort = value.get_numb(0);
	value.Sort(funct);                                 /** Виклик метода сортування */
	int afterSort = value.get_numb(0);
```

```
	if (beforeSort != afterSort && afterSort > value.get_numb(2))
		cout << "Тест функции сортировки списка\t\t выполнен успешно." << endl <<
endl;
	else
		cout << "Тест функции сортировки списка\t\t не выполнен успешно." << endl <<
endl;
}

void FuncTester::Task_Test()				/** Реалізація тестування метода
реалізації індивідуального завдання */
{
	if (value.Task() == 8)				/** Виклик метода реалізації
індивідуального завдання */
		cout << endl << "Тест функции индивидуального задания\t выполнен успешно." <<
endl;
	else
		cout << endl << "Тест функции индивидуального задания\t не выполнен успешно."
<< endl;
}

FuncTester::FuncTester() {}				/** Реалізація конструктора за
замовчуванням */
    FuncTester::~FuncTester() {}			/** Реалізація деструктора */
```

Functor.cpp

```
/**
* @file Functor.cpp
* Файл реалізації методів класу Functor
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#include "Functor.h" /** Підключення файлу Functor.h */

bool Functor::operator()(const unique_ptr<Student>& first, const unique_ptr<Student>&
second) const /** Реалізація перегруження оператору () */
{
	if (choise == 1) /** Якщо треба сортувати по назві */
	{
		if (direction == true) /** Вибір напряму сортування */
			return first->get_name() > second->get_name();
		else
			return first->get_name() < second->get_name();
	}
	else if (choise == 2) /** Якщо треба сортувати по номеру */
	{
		if (direction == true) /** Вибір напряму сортування */
			return first->get_number() > second->get_number();
		else
			return first->get_number() < second->get_number();
	}
	else if (choise == 3) /** Якщо треба сортувати по середньому балу */
	{
		if (direction == true) /** Вибір напряму сортування */
			return first->get_mark() > second->get_mark();
		else
			return first->get_mark() < second->get_mark();
	}
	else if (choise == 4) /** Якщо треба сортувати по боргу з програмування */
	{
		if (direction == true) /** Вибір напряму сортування */
			return first->get_prog_d() > second->get_prog_d();
		else
			return first->get_prog_d() < second->get_prog_d();
```

```
	}
	else if (choise == 5) /** Якщо треба сортувати по віку */
	{
		if (direction == true) /** Вибір напряму сортування */
			return first->get_age() > second->get_age();
		else
			return first->get_age() < second->get_age();
	}
	else if (choise == 6) /** Якщо треба сортувати по боргу */
	{
		if (direction == true) /** Вибір напряму сортування */
			return first->get_debt() > second->get_debt();
		else
			return first->get_debt() < second->get_debt();
	}
}

Functor::Functor(bool direction, int choise) :direction(direction), choise(choise) {} /**
Реалізація конструктора з параметрами */
		Functor::~Functor() {} /** Реалізація деструктора */
```

list.cpp

```
/**
* @file list.cpp
* Файл реалізації методів класу CList
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#include "list.h" /** Підключення файлу list.h */

CList::~CList() /** Реалізація деструктора */
{
	StudentList.clear(); /** Очищення пам'яті масива програм */
}

CList::CList() /** Реалізація конструктора за замовчуванням */
{
	for (size_t i = 0; i < 5; i++) /** Масив, який заповнює вектор елементами */
	{
		if (i == 0)
			StudentList.emplace_back(new Student());
		else if (i == 1)
			StudentList.emplace_back(new Course(18, 2, 8, "Andry", 1, 20, 2));
		else if (i == 2)
			StudentList.emplace_back(new Student(19, 3, 7, "Jhon", 1, 30));
		else if (i == 3)
			StudentList.emplace_back(new Surnames(19, 4, 10, "Dmitry", 0, 0,
"Miler"));
		else
			StudentList.emplace_back(new Course(20, 5, 7, "Jim", 1, 30, 3));
	}
}

void CList::PrintList() const noexcept /** Реалізація методу виведення списку на екран */
{
	try
	{
		if (StudentList.size() == 0) /** Перевірка розміру списка */
			throw exception("Список пуст.");

		int value = 1;

		cout << endl << setiosflags(ios::left);
		cout << setw(10) << "Возраст" << setw(8) << "Номер";
```

```
		cout << setw(15) << "Средний балл" << setw(7) << "Имя";
		cout << setw(10) << "Долг" << setw(14) << "Долг(прог.)";
		cout << setw(7) << "Курс/Староста" << endl; /** Виведення назв полів класів */

		for_each(StudentList.begin(), StudentList.end(), [&value](const
unique_ptr<Student>& Student) /** Цикл, який виводить список елементів */
		{
			cout << *Student << endl; /** Виведення номеру та полів елемента */
			value++; /** Збільшення зміної нумерування */
		});

		cout << endl;
	}
	catch (const std::exception& ex)
	{
		cout << ex.what() << endl;
	}
}

int CList::Task() const /** Реалізація метода виконання індивідуального завдання */
{
	int value = 1; /** Оголошення змінної кількості відповідних елементів */

	for (const auto& Student : StudentList) /** Цикл який обходить усі елементи масиву */
	{
		if (Student->get_debt() == 1) /** Перевірка полів елементів */
		{
			cout << value << ". " << *Student << endl; /** Виведення елемента на
екран */
			value++; /** Збільшення змінної нумерування масиву */
		}
	}

	if (value == 0) /** Якщо елементів немає */
		throw exception("Программы с такими параметрами отсутствуют.");

	return value; /** Повернення кількості відповідних елементів */
}

int CList::AddStudent(int value) /** Реалізація метода додавання нової програми */
{
	if (value == 1)
	{
		Student* temp = new Student; /** Створення елемента класу Student */
		StudentList.emplace_back(temp); /** Вставка елемента у масив */
	}
	else if (value == 2)
	{
		Student* temp = new Course; /** Створення елемента класу Course */
		StudentList.emplace_back(temp); /** Вставка елемента у масив */
	}
	else if (value == 3)
	{
		Student* temp = new Surnames; /** Створення елемента класу Surnames */
		StudentList.emplace_back(temp); /** Вставка елемента у масив */
	}
	else
		throw exception("Неверная команда.");

	return StudentList.size(); /** Повернення розміру списку */
}

int CList::DeleteStudent(int value) /** Реалізація метода видалення програми */
{
	if (StudentList.size() == 0) /** Перевірка розміру списку */
		throw exception("Список программ пуст.");
```

```
	int number = -1;
	bool findEl = false;
	std::vector<unique_ptr<Student>>::iterator it; /** Створення ітератора */

	for (const auto& Student : StudentList) /** Цикл, який обробляє усі елементи колекції
*/
	{
		if (Student->get_number() == value) /** Якщо програма має потрібний індекс */
		{
			number++; /** Збільшення змінної номеру елементу */
			findEl = true; /** Змінення зміної знаходження елемента */
			break; /** Зупинка роботи циклу */
		}
		else
			number++; /** Збільшення змінної номеру елементу */
	}

	if (findEl) /** Якщо елемент з потрібним індексом був найден */
	{
		it = StudentList.begin(); /** Встановлення покажчика на початок списку */
		advance(it, number); /** Переміщення покажчика на потрібний елемент */
		StudentList.erase(it); /** Видалення потрібного елементу */

		cout << "Удаление выполнено." << endl;
	}
	else
		throw exception("Элемент не найден.");

	return StudentList.size(); /** Повернення змінної розміру списку */
}

void CList::Sort(Functor& choise) noexcept /** Реалізація метода сортування списку */
{
	std::sort(StudentList.begin(), StudentList.end(), choise); /** Сортування списку */
}

int CList::SaveFile(string filename) const /** Реалізація метода запису даних у файл */
{
	if (StudentList.size() == 0) /** Перевірка розміру масиву */
		throw exception("Список пуст.");

	ofstream fout(filename); /** Відкриття файлу */
	if (!fout.is_open()) /** Перевірка чи відкритий файл */
		throw exception("Невозможно открыть файл.");

	fout << setiosflags(ios::left);
	fout << setw(10) << "Возраст" << setw(8) << "Номер";
	fout << setw(15) << "Средний балл" << setw(7) << "Имя";
	fout << setw(10) << "Долг" << setw(14) << "Долг(прог.)";
	fout << setw(7) << "Курс/Староста" << endl; /** Виведення назв полів */

	int value = 1;
	for (const auto& Student : StudentList) /** Цикл, який обробляє усі елементи масиву
*/
	{
		fout << setiosflags(ios::left) << setw(2) << value << ". "; /** Виведення
нумерування елементів */
		fout << *Student << endl; /** Виведення полів елементів */
		value++;
	}

	fout.close(); /** Закриття файлу */
	return value - 1; /** Повернення кількості елементів, які були збережені у файл */
}
```

```
int CList::ReadFile(string filename) /** Реалізація метода читання даних з файлу */
{
	regex expressionStudent("([\\d]* [\\d]* [\\d]* [A-Z]+[\\w,.;:-]* [0|1] [\\d]* )");
/** Регулярний вираз для класу Student */
	regex expressionCourse("([\\d]* [\\d]* [\\d]* [A-Z]+[\\w,.;:-]* [0|1] [\\d]* [\\d]*
)"); /** Регулярний вираз для класу Course */
	regex expressionSurnames("([\\d]* [\\d]* [\\d]* [A-Z]+[\\w,.;:-]* [0|1] [\\d]* [A-
Z]+[\\w,.;:-]* )"); /** Регулярний вираз для класу Surnames */
	regex replaceSymbols("[;:-]"); /** Регулярний вираз для символів, які треба замінити
*/

	string temp, data, replacement = "";
	int value = 0;
	ifstream fin(filename); /** Відкриття файлу */
	istringstream ss;

	if (!fin.is_open()) /** Перевірка чи відкритий файл */
	{
		throw exception("Невозможно открыть файл для чтения.");
		return StudentList.size();
	}

	while (!fin.eof()) /** Цикл, який працює до кінця файлу */
	{
		getline(fin, data); /** Отримання строки з файлу */
		temp = regex_replace(data, replaceSymbols, replacement); /** Видалення
символів які не підходять */
		temp += " ";

		if (regex_match(temp, expressionStudent)) /** Перевірка на відповідність до
класу Student */
		{
			istringstream ss(temp);
			Student* Stud = new Student(); /** Створення змінної класу Student */

			ss >> *Stud;
			StudentList.emplace_back(Stud); /** Вставка елемента у масив */
			value++; /** Збільшення змінної кількості нових елементів */
		}
		else if (regex_match(temp, expressionCourse)) /** Перевірка на відповідність
до класу Student */
		{
			istringstream ss(temp);
			Course* Student = new Course(); /** Створення змінної класу Course */

			ss >> *Student;
			StudentList.emplace_back(Student); /** Вставка елемента у масив */
			value++; /** Збільшення змінної кількості нових елементів */
		}
		else if (regex_match(temp, expressionSurnames)) /** Перевірка на відповідність
до класу Student */
		{
			istringstream ss(temp);
			Surnames* Student = new Surnames(); /** Створення змінної класу
Surnames */

			ss >> *Student;
			StudentList.emplace_back(Student); /** Вставка елемента у масив */
			value++; /** Збільшення змінної кількості нових елементів */
		}
	}

	fin.close(); /** Закриття файлу */
	return value; /** Повернення змінної кількості нових елементів */
}
```

```
int CList::get_numb(int value) const /** Реалізація метода отримання номеру елементів */
{
	return StudentList[value]->get_number(); /** Повернення номеру */
}
```

main.cpp

```
/**
* @mainpage
* <b> Розрахункове завдання. <br/></b>
* <b><i> Індивідуальне завдання: Визначити, хто із студентів за результатами сесії має заборгованості і який % становлять їх заборгованості з програмування </i></b><br/>
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

/**
* @file main.cpp
* Файл реалізації методів класу Menu
*/

#include "list.h" /** Підключення файлу list.h */
#include "Menu.h" /** Підключення файлу Menu.h */
#include "FuncTester.h" /** Підключення файлу FuncTester.h */

void main()
{
	setlocale(LC_ALL, "ru"); /** Русифікація консолі */
	Menu menu; /** Створення змінної класу Menu */

	menu.menu(); /** Виклик методу меню */

	cout << endl << "Тестирование функций с помощью класса-тестировщика" << endl;
	FuncTester tester; /** Створення змінної класу FuncTester */
	tester.ReadFile_Test(); /** Виклик методу тестування методу читання файлу */
	tester.Add_Test(); /** Виклик методу тестування методу додавання елементів */
	tester.Delete_Test(); /** Виклик методу тестування методу видалення елементів */
	tester.SaveFile_Test(); /** Виклик методу тестування методу виведення даних у файл */
	tester.Sort_Test(); /** Виклик методу тестування методу сортування */
	tester.Task_Test(); /** Виклик методу тестування методу виконання індивідуального завдання */
	tester.~FuncTester();

	if (_CrtDumpMemoryLeaks()) /** Перевірка витоків пам'яті */
		cout << endl << "Есть утечка памяти." << endl; /** Виведення повідомленя якщо витоки є */
	else
		cout << endl << "Утечка памяти отсутствует." << endl; /** Виведення повідомленя якщо витоків немає */

	system("PAUSE"); /** Зупинка роботи програми */
	return; /** Завершення роботи програми */
}
```

Menu.cpp

```
/**
* @file Menu.cpp
* Файл реалізації методів класу Menu
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#include "Menu.h" /** Підключення файлу Menu.h*/

void Menu::menu() const /** Виклик метода діалогового меню*/
{
```

```
	/** Оголошення змінних для роботи студента*/
	int choise, choise2;
	int size = -1, value;
	bool stop = true;
	string filename; /** Назва файлу */
	CList list; /** Список елементів */

	while (stop != 0) /** Цикл, який працює поки користувач не захоче вийти*/
	{
		cout << endl;
		cout << "1) Вывести список на экран" << endl;
		cout << "2) Удаление элемента" << endl;
		cout << "3) Добавление элемента" << endl;
		cout << "4) Индивидуальное задание" << endl;
		cout << "5) Работа с файлами" << endl;
		cout << "6) Сортировка" << endl;
		cout << "7) Завершение работы" << endl;
		cout << "=========================" << endl;
		cout << "Ваш выбор: ";
		cin >> choise; /** Введення зміної вибору дії*/
		cout << endl;

		switch (choise)
		{
		case 1:

			list.PrintList(); /** Виклик методу виведення списку у консоль */

			break;

		case 2:
			cout << "Введите номер элемента, который хотите удалить: ";
			cin >> choise; /** Введення зміної вибору дії*/
			cout << endl;

			try
			{
				list.DeleteStudent(choise); /** Виклик методу видалення елемента
*/
			}
			catch (const std::exception& ex)
			{
				cout << ex.what() << endl;
			}

			break;

		case 3:
			cout << "Выберите программу, которую хотите добавить:" << endl;
			cout << "1. Элемент класса Student" << endl;
			cout << "2. Элемент класса Course" << endl;
			cout << "3. Элемент класса Surnames" << endl;
			cout << "==============================" << endl;
			cout << "Ваш выбор: ";
			cin >> value; /** Введення зміної вибору дії*/

			try
			{
				list.AddStudent(value); /** Виклик методу додавання студента */
			}
			catch (const std::exception & ex)
			{
				cout << ex.what() << endl;
			}
			cout << "Программа добавлена." << endl;
```

```
                        break;
                case 4:
                        cout << "Индивидуальное задание:\nУзнать, кто из студентов имеет
задолжности, и какой процент задолжности по программированию" << endl;

                        try
                        {
                                list.Task(); /** Виклик методу виконання індивідуального
завдання */
                        }
                        catch (const std::exception& ex)
                        {
                                cout << ex.what() << endl;
                        }

                        break;

                case 5:
                        cout << "Что делать?" << endl;
                        cout << "1) Запись в файл" << endl;
                        cout << "2) Чтение файла" << endl;
                        cout << "3) Вернуться назад" << endl;
                        cout << "==================" << endl;
                        cout << "Ваш выбор: ";
                        cin >> choise2; /** Введення зміної вибору дії*/
                        cout << endl;

                        if (choise2 == 1)
                        {
                                string::size_type n;
                                cout << "Введите название файла для записи: ";
                                cin >> filename; /** Введення назви файлу */
                                cout << endl;

                                n = filename.find(".txt");
                                if (n > 187) filename += string(".txt");

                                try
                                {
                                        list.SaveFile(filename); /** Виклик методу запису списка
у файл */
                                }
                                catch (const std::exception & ex)
                                {
                                        cout << ex.what() << endl;
                                }

                                cout << "Запись завершена. " << endl;
                        }
                        else if (choise2 == 2)
                        {
                                string::size_type n;
                                cout << "Введите название файла для чтения: ";
                                cin >> filename; /** Введення назви файлу*/
                                cout << endl;

                                n = filename.find(".txt");
                                if (n > 187) filename += string(".txt");

                                try
                                {
                                        list.ReadFile(filename); /** Виклик методу читання даних
з файлу */
                                }
                                catch (const std::exception & ex)
                                {
```

```
                                cout << ex.what() << endl;
                        }

                        cout << "Чтение файла завершено." << endl;

                }
                else if (choise2 == 3)
                        cout << "Возвращение назад. " << endl;
                else
                        cout << "Неверная команда. Повторите попытку." << endl;

                break;

        case 6:
                cout << "Сортировать по: " << endl;
                cout << "1) Возрастанию" << endl;
                cout << "2) Убыванию" << endl;
                cout << "3) Вернуться назад" << endl;
                cout << "==================" << endl;
                cout << "Ваш выбор: ";
                cin >> choise2; /** Введення зміної напряму сортування*/
                cout << endl;

                if (choise2 == 1 || choise2 == 2)
                {
                        cout << "По какому полю сортировать: " << endl;
                        cout << "1) Название" << endl;
                        cout << "2) Индекс" << endl;
                        cout << "3) Количество строк кода" << endl;
                        cout << "4) Размер программы" << endl;
                        cout << "5) Время выполнения кода" << endl;
                        cout << "6) Использует ли программа интернет или нет" << endl;
                        cout << "============================" << endl;
                        cout << "Ваш выбор: ";
                        cin >> value;
                        cout << endl;

                        if (value > 0 && value <= 6)
                        {
                                Functor funct(choise2 - 1, value); /** Оголошення змінної
класу Functor */
                                list.Sort(funct); /** Виклик методу сортування */

                                cout << "Список отсортирован. " << endl;
                        }
                        else
                                cout << "Неверный символ." << endl;
                }
                else if (choise2 == 3)
                        cout << "Возвращение назад." << endl;
                else
                        cout << "Ошибка. Неверная команда." << endl;

                break;
        case 7:
                cout << "Завершение работы." << endl;
                stop = false;
                break;

        default:
                cout << "Неверный символ. Повторите попытку." << endl;
                break;

        }

    }
```

```
	return; /** Завершення роботи метода */
}

Menu::Menu() {} /** Реалізація конструктора за замовчуванням */
                        Menu::~Menu() {} /** Реалізація деструктора */
```

Student.cpp

```
/**
* @file Student.cpp
* Файл реалізації методів базового класу
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#include "Student.h"

int Student::get_age() const /** Реалізація геттера часу віку студента */
{
	return age; /** Повернення віку студента */
}

int Student::get_prog_d() const /** Реалізація геттера боргу з програмування */
{
	return prog_d; /** Повернення боргу з програмування */
}

int Student::get_mark() const /** Реалізація геттера середньої оцінки */
{
	return middle_mark; /** Повернення середньої оцінки */
}

int Student::get_number() const /** Реалізація геттера номеру студента */
{
	return number_stud; /** Повернення номера студента */
}

bool Student::get_debt()const /** Реалізація геттера боргу студента */
{
	return debt; /** Повернення боргу студента */
}

string Student::get_name()const /** Реалізація геттера імені студента */
{
	return name; /** Повернення імені студента */
}

void Student::enter(istream& data) /** Реалізація перевантаженого оператора вводу */
{
	data >> age >> number_stud >> middle_mark >> name >> debt >> prog_d; /** Введення
даних у об'єкт типу istream */
}

string Student::getInfo() const /** Реалізація методу отримання даних програми */
{
	stringstream temp; /** Створення змінної типу stringstream */
	temp.setf(ios::left);

	temp << setw(10) << age << setw(8) << number_stud << setw(16) << middle_mark <<
setw(9)
		<< name << setw(7) << debt << setw(14) << prog_d; /** Запис даних у об'єкт
типу stringstream */

	return temp.str(); /** Повернення даних у форматі string */
}
```

```
Student::Student(int a, int n, int m, string na, bool d, int pd) : age(a), number_stud(n), middle_mark(m), name(na), debt(d), prog_d(pd) {} /** Реалізація конструктора з параметрами */
Student::Student() : age(17), number_stud(1), middle_mark(8), name("Bond"), debt(1), prog_d(15) {} /** Реалізація конструктора за замовчуванням */
Student::Student(const Student& other) : age(other.age), number_stud(other.number_stud), middle_mark(other.middle_mark), name(other.name), debt(other.debt), prog_d(other.prog_d) {} /** Реалізація конструктора копіювання */
Student::~Student() {} /** Реалізація деструктора */

ofstream& operator<< (ofstream& output, const Student& Student) /** Реалізація перевантаженого оператора запису даних у файл */
{
	output << Student.getInfo(); /** Виклик фкнції отримання інформації */
	return output; /** Повернення інформації */
}

ostream& operator<< (ostream& output, const Student& Student) /** Реалізація перевантаженого оператора виводу даних у консоль */
{
	output << Student.getInfo(); /** Виклик фкнції отримання інформації */
	return output; /** Повернення інформації */
}

istream& operator>> (istream& input, Student& Student) /** Реалізація перевантаженого оператора вводу даних з консолі */
{
	Student.enter(input); /** Виклик метода вводу даних */
	return input; /** Повернення інформації */
}

bool Student::operator!=(const string type) const /** Реалізація перевантаженого оператора нерівності */
{
	return true; /** Перевірка відбувається за типом зловмисного ПО (звичайна програма не може бути зловмисним ПО) */
}

bool Student::operator==(const int number_stud) const /** Реалізація перевантаженого оператора порівняння */
{
	return this->number_stud == number_stud; /** Перевірка відбувається за номером */
}

Student& Student::operator= (Student& temp) /** Реалізація перевантаженого оператора присвоювання */
{
	if (this == &temp) /** Перевірка якщо змінні однакові */
		return *this;

	int age = temp.age; /** Присвоювання поля віку */
	int prog_d = temp.prog_d; /** Присвоювання поля боргу з програмування */
	int middle_mark = temp.middle_mark; /** Присвоювання поля середньої оцінки */
	int number_stud = temp.number_stud; /** Присвоювання поля номера студента */
	bool debt = temp.debt; /** Присвоювання поля боргу */
	string name = temp.name; /** Присвоювання поля імені студента */

	return *this; /** Повернення програми */
}
```

Surnames.cpp

```
/**
* @file Surnames.cpp
```

```
* Файл реалізації методів класу-спадкоємця
* @author Tatarenko Andrey
* @date 2020.06.16
* @version 1.0
*/

#include "Surnames.h" /** Підключення файлу Surnames.h */

int Surnames::get_age() const /** Реалізація геттера часу віку студента */
{
	return age; /** Повернення віку студента */
}

int Surnames::get_prog_d() const /** Реалізація геттера боргу з програмування */
{
	return prog_d; /** Повернення боргу з програмування */
}

int Surnames::get_mark() const /** Реалізація геттера середньої оцінки */
{
	return middle_mark; /** Повернення середньої оцінки */
}

int Surnames::get_number() const /** Реалізація геттера номеру студента */
{
	return number_stud; /** Повернення номера студента */
}

bool Surnames::get_debt()const /** Реалізація геттера боргу студента */
{
	return debt; /** Повернення боргу студента */
}

string Surnames::get_name()const /** Реалізація геттера імені студента */
{
	return name; /** Повернення імені студента */
}

string Surnames::getInfo() const /** Реалізація методу отримання даних програми */
{
	stringstream temp; /** Створення змінної типу stringstream */
	temp.setf(ios::left);

	temp << setw(10) << age << setw(8) << number_stud << setw(16) << middle_mark <<
setw(9)
		<< name << setw(7) << debt << setw(14) << prog_d << setw(10) << star; /**
Запис даних у об'єкт типу stringstream */

	return temp.str(); /** Повернення даних у форматі string */
}
void Surnames::enter(istream& data) /** Реалізація перевантаженого оператора вводу */
{
	data >> age >> number_stud >> middle_mark >> name >> debt >> prog_d >> star; /**
Введення даних у об'єкт типу istream */
}

Surnames::Surnames(int a, int n, int m, string na, bool d, int pd, string st) : Student(a,
n, m, na, d, pd), star(st) {}
Surnames::Surnames() : Student(), star("Petrov") {}
Surnames::Surnames(const Surnames& other) : Student(other), star(other.star) {}
Surnames::~Surnames() {}

Surnames& Surnames::operator=(Surnames& temp) /** Реалізація перевантаженого оператора
присвоювання */
{
	if (this == &temp) /** Перевірка якщо змінні однакові */
```

```
                return *this;

        Surnames::operator=(temp); /** Присвоювання полів базового класу */
        string star = temp.star; /** Присвоювання прізвища старости */

        return *this;
}

bool Surnames::operator!=(const string type) const /** Реалізація перевантаженого оператора
нерівності */
{
        return true; /** Перевірка відбувається за типом зловмисного ПО (звичайна програма не
може бути зловмисним ПО) */
}

bool Surnames::operator==(const int number_stud) const /** Реалізація перевантаженого
оператора порівняння */
{
        return this->number_stud == number_stud; /** Перевірка відбувається за номером */
```

Додаток Б

Результати роботи програми

```
1) Вывести список на экран
2) Удаление элемента
3) Добавление элемента
4) Индивидуальное задание
5) Работа с файлами
6) Сортировка
7) Завершение работы
=========================
Ваш выбор: 7

Завершение работы.

Тестирование функций с помощью класса-тестировщика
Тест функции чтения файла                выполнен успешно.
Тест функции добавления студента         выполнен успешно.
Удаление выполнено.
Тест функции удаления студента           выполнен успешно.
Тест функции сохранения в файл           выполнен успешно.
Тест функции сортировки списка           выполнен успешно.

1. 20        10      4              Dima     1      30
2. 17        9       7              Adolf    1      25            Petrov
3. 18        6       6              Mark     1      15
4. 19        3       7              Jhon     1      30
5. 18        2       8              Andry    1      20            2
6. 17        1       8              Bond     1      15
7. 17        1       8              Bond     1      15            1

Тест функции индивидуального задания     выполнен успешно.

Утечка памяти отсутствует.
Press any key to continue . . .
```

Рис. 10. Результати роботи програми