

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Системы автоматизированного проектирования | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Программирование | | |
| Курс | I | | | Группа | 423 |

Отчёт по контрольной работе №2

Вариант № 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 423 |  |  |  | Ефремов Иван Андреевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Макарук Роман Валерьевич |
|  |  |  |  | Федин Алексей Константинович |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc132966445)

[2 Исходные данные 3](#_Toc132966446)

[3 Особые ситуации 3](#_Toc132966447)

[4 Математические методы и алгоритмы решения задач 4](#_Toc132966448)

[5 Форматы представления данных 4](#_Toc132966449)

[6 Структура программы 5](#_Toc132966450)

[7 Описание хода выполнения 9](#_Toc132966451)

[8 Блок-схема алгоритма решения задачи 9](#_Toc132966452)

[9 Результаты работы программы 11](#_Toc132966453)

[10 Выводы по заданию 13](#_Toc132966454)

[11 Исходный код полученного программного решения 13](#_Toc132966455)

## 1 Постановка задачи

Задан базовый и производный класс. Разработать поля и методы, наследуемые из базового класса, а также собственные. Компоненты производных классов. Базовый класс может быть абстрактным. Реализовать возможность получения списка объектов в контейнере.

## 2 Исходные данные

Исходные данные состоят из фамилии, имени, адреса, университета, курса и номера группы студента. Данные могут быть введены пользователем с консоли или из файла.

## 3 Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

-если пользователь ввёл количество студентов меньше 1, то программа попросит его ввести данные хотя бы об одном студенте;

-если пользователь ввёл номер группы меньше 1, то программа попросит ввести номер группы как натуральное число;

-если при запросе числа пользователь ввёл другие символы, то программа укажет на необходимость ввода целых чисел и попросит повторить ввод;

-если при запросе выбрать пункт меню пользователь введет несуществующий вариант, то программа укажет, что такого пункта меню нет, выведет возможные для выбора пункты меню и попросит пользователя ввести номер пункта еще раз;

-если при попытке отфильтровать студентов пользователь выберет такие условия, под которые не подходит ни один студент, программа сообщит об этом;

-если после предложения пользователю сохранить отфильтрованные результаты в файл, пользователь введет что-то кроме “n” или “y”, то программа сообщит, какой должен быть формат ввода и попросит пользователя ввести свой выбор еще раз;

-если пользователь попытается открыть несуществующий файл для чтения, программа сообщит, что такого файла не существует, и попросит пользователя указать файл для чтения еще раз;

-если пользователь выбрал ввод с файла и в одной из строк файла будет указано неверное количество полей класса, программа сообщит об этом и попросит пользователя открыть новый файл или изменить данные в этом файле;

-если пользователь выбрал ввод с файла и в одной из строк значение полей «курс» или «номер группы» не будет целым числом или будет меньше одного, то программа сообщит об этом и попросит пользователя открыть новый файл или изменить данные в этом файле;

## 4 Математические методы и алгоритмы решения задач

Согласно постановке задачи для получения необходимых списков будут использоваться операторы “>”, “<” и “==”.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 Форматы представления данных Таблица 1 - Основные переменные, используемые в программе   |  |  |  | | --- | --- | --- | | Имя | Тип | Описание | | actionTop | int | Выбор пользователя в меню верхнего уровня | | actionBottom | int | Выбор пользователя в меню нижнего уровня | | students | Vector<Student> | Вектор с данными о студентах | | numberOfStudents | Unsigned int | Количество студентов |   Таблица 2 - Класс Patient   |  |  |  |  | | --- | --- | --- | --- | | Имя | Описание | Модификатор | Тип | | name | Имя студента | private | string | | surname | Фамилия студента | private | string | | address | Адрес студента | private | string | | university | Название университета | private | string | | course | Курс студента | private | Unsigned int | | group | Группа студента | private | Unsigned int |   Данные, записанные в файле для чтения, должны быть представлены таким образом:  Имя1 Фамилия1 Адрес1 Университет1 Курс1 Группа1  .  .  .  Имяn Фамилияn Адресn Университет n Курс n Группаn  Данные о каждом студенте должны быть расположены на разных строках, причем каждое поле класса должно быть разделено пробелом.  Данные в файле вывода записываются в формате:  «Условие фильтрации» «Ключ фильтрации»:  Information about student #1:  Name:  Surname:  Adress:  University:  Course:  Group:  .  .  .  Information about student #n:  Name:  Surname:  Adress:  University:  Course:  Group: 6 Структура программы Таблица 3 - Модули программы   |  |  | | --- | --- | | Название модуля | Описание | | Start | Запуск основных функций программы | | UserInterface | Зацикленное двухуровневое меню, открытие файлов, вывод и запись необходимой информации в консоль и в файл | | GetInfo | Функции для считывания данных с консоли, а также подключение основных библиотек | | DataInput | Заполнение массива с консоли или из файла, разделение строк по пробелам и сортировка массива по возрастанию номера карты | | Tasks | Фильтрация данных из массива по выбору пользователя, функция для сравнения переменных класса string без учёта регистра | | Person | Базовый класс | | Student | Класс-наследник | | ModuleTests | Модульные тесты для проверки корректности работы функций программы |   Таблица 4 - Функции класса Person   |  |  | | --- | --- | | Имя | Описание | | SetInformation | Установить значение в поля класса | | GetSurname | Передать фамилию студента | | GetAdress | Передать адресс студента |   Таблица 5 - Функции класса Student   |  |  | | --- | --- | | Имя | Описание | | ShowInformation | Вывести данные о студенте | | SetInformation | Установить значение в поля класса | | GetAll | Передать все данные о студенте | | GetName | Передать имя студента | | GetUniversity | Передать название университета | | GetCourse | Передать курс студента | | GetGroup | Передать группу студента | |  |
|  |  |

Таблица 6 – Основные функции модуля UserInterface

|  |  |
| --- | --- |
| Имя | Описание |
| StartProgram | Зацикленное меню верхнего уровня, в котором находятся основные функции программы |
| WorkWithConsole | Заполнение массива данными из консоли и вывод результата фильтрации, выбранной пользователем, в консоль и, по выбору пользователя, в файл |
| WorkWithFile | Заполнение массива данными из файла и вывод результата фильтрации, выбранной пользователем, в файл |
| OpenFile | Открытие файла для чтения или записи, по выбору пользователя |
| SaveResults | Предложение пользователю сохранить результат фильтрации в файл |

Таблица 7 – Функции модуля GetInfo

|  |  |
| --- | --- |
| Имя | Описание |
| IgnoreLine | При неверном вводе пользователя очищает поток ввода |
| GetInt | Считывание целого числа из консоли |
| GetString | Считывание строки с консоли |
| GetLink | Считывание пути к файлу с консоли |
| GetChar | Считывание ответа пользователя в формате y/n |

Таблица 8 – Функции модуля DataInput

|  |  |
| --- | --- |
| Имя | Описание |
| SplitString | Разделение строки по пробелам |
| ConsoleInput | Ввод данных с консоли и заполнение вектора |
| FileInput | Ввод данных из файла и заполнение вектора |

Таблица 9 – Функции модуля Tasks

|  |  |
| --- | --- |
| Имя | Описание |
| Comparator | Компаратор, задающий условия сортировки |
| Sort | Сортировка вектора |
| Comparison | Сравнение переменных класса string без учета регистра символов |
| FilterByUniversity | Фильтр студентов по университету, выбранному пользователем |
| FilterByCourse | Фильтр студентов по курсу, выбранному пользователем |
| FilterByGroup | Фильтр студентов по группе, выбранной пользователем |
| Print | Вывод результата фильтрации в консоль или в файл, по выбору заполнения вектора |

Таблица 10 – Функции модуля ModuleTests

|  |  |
| --- | --- |
| Имя | Описание |
| Check | Проверка соответствия считанных данных с данными из программы |
| Test1 | Проверка корректности ввода данных с файла |
| Test2 | Проверка корректности работы фильтра по университету |
| Test3 | Проверка корректности работы фильтра по номеру группы |
| Test4 | Проверка корректности подсчёта строк в файле |
| Test5 | Проверка корректности работы считывания данных с консоли |
| LaunchAllTests | Запуск всех тестов |

## 7 Блок-схема алгоритма решения задачи

На рисунке 1 представлена блок-схема алгоритма решения задачи.



Рисунок 1 – Блок-схема алгоритма решения задачи

## 8 Описание хода выполнения

Для выполнения задания необходимо изучить наследование классов в языке C++.

После обучения этому приступили к реализации основных задач программы – созданию базового класса и класса-наследника, заполнению полей и разбиению структуры программы на модули.

Во время работы программа предлагает пользователю три варианта фильтрации студентов (по университету, по курсу и по группе) и вывод информации о всех студентах на экран. Если пользователь выбрал один из вариантов фильтрации программа запрашивает от пользователя необходимое ему условие фильтрации. После чего программа перебором отбирает студентов, соответствующих данному условию, и выводит информацию о них на экран.

## 9 Результаты работы программы

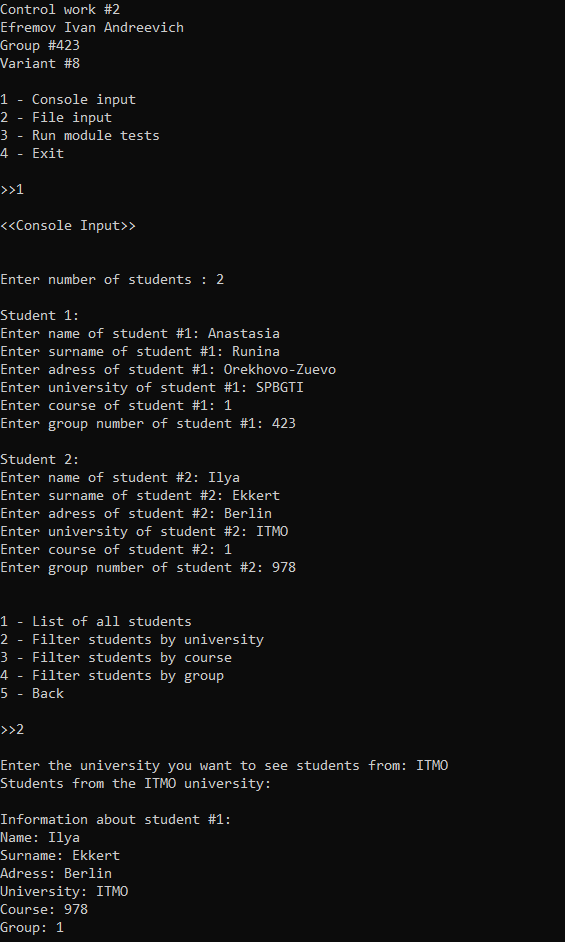


Рисунок 2 – Экранная копия результата работы разработанной программы

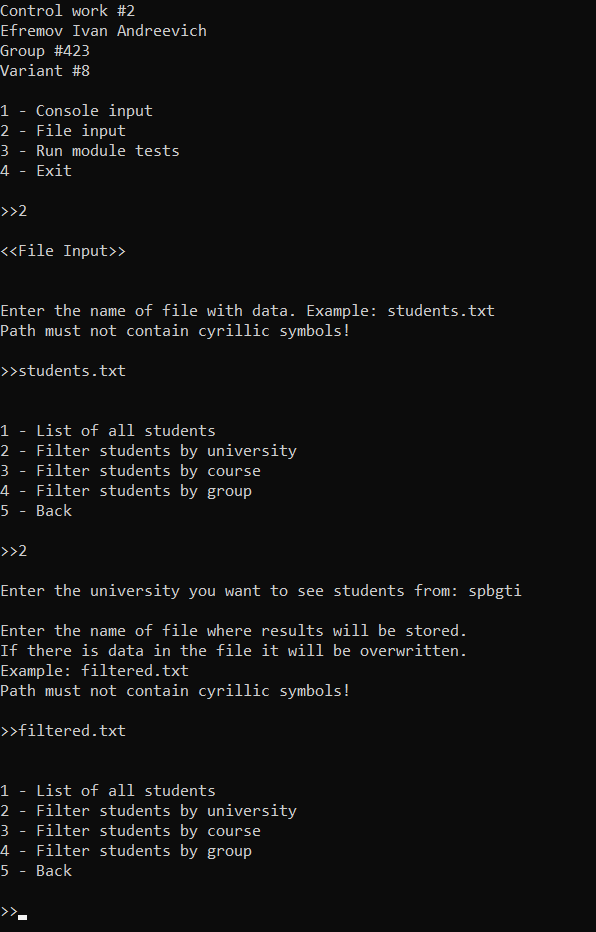


Рисунок 3 – Экранная копия результата работы разработанной программы

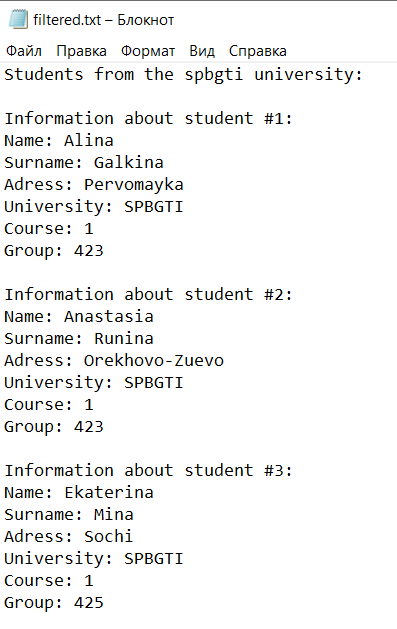


Рисунок 4 – Результат работы программы, записанный в файл

## 10 Выводы по заданию

В ходе выполнения задания были изучены методы работы с языком С++ и реализацией классов, а также были получены навыки работы с многофайловыми проектами. Были устранены недостатки в знаниях библиотек и получены новые навыки программирования.

## 11 Исходный код полученного программного решения

Файл Start:

#include "UserInterface.h"

int main() { StartProgram(); }

Файл UserInterface.h:

#pragma once

#include "Student.h"

#include <fstream>

#include <vector>

enum WorkWithFiles { input, output };

enum TopMenu { console = 1, file, module, quit };

enum BottomMenu { all = 1, university, course, group, back };

void WorkWithConsole(vector<Student>& students, int& actionBottom);

void WorkWithFile(vector<Student>& students, int& actionBottom);

void Fio();

void OptionsTop();

void InputOption(int option);

void OptionsBottom();

void IncorrectOption();

char SaveResults(fstream& fout);

string OpenFile(int option, fstream& file);

void StartProgram();

Файл UserInterface.cpp:

#include "UserInterface.h"

#include "Tasks.h"

#include "DataInput.h"

#include "ModuleTests.h"

//чтение данных с консоли и работа с ними

void WorkWithConsole(vector<Student>& students, int& actionBottom)

{

ui numberOfStudents = 0;

actionBottom = 1;

ConsoleInput(students, numberOfStudents);

while (actionBottom != BottomMenu::back) {

OptionsBottom();

actionBottom = GetInt(">>");

switch (actionBottom)

{

case BottomMenu::all:

Print(TopMenu::console, students, "All students we have data about:\n\n");

break;

case BottomMenu::university:

FilterByUniversity(TopMenu::console, numberOfStudents, students);

break;

case BottomMenu::course:

FilterByCourse(TopMenu::console, numberOfStudents, students);

break;

case BottomMenu::group:

FilterByGroup(TopMenu::console, numberOfStudents, students);

break;

case BottomMenu::back:

system("cls");

Fio();

break;

default:

IncorrectOption();

break;

}

}

}

//чтение данных из файла и работа с ними

void WorkWithFile(vector<Student>& students, int& actionBottom)

{

ui numberOfStudents = 0;

fstream docIn, docOut;

bool isCorrect = false;

do {

string inputName = OpenFile(WorkWithFiles::input, docIn);

isCorrect = FileInput(students, docIn, numberOfStudents, inputName);

docIn.close();

} while (isCorrect == false);

while (actionBottom != BottomMenu::back) {

OptionsBottom();

actionBottom = GetInt(">>");

switch (actionBottom)

{

case BottomMenu::all:

Print(TopMenu::file, students, "All students we have data about:\n\n");

break;

case BottomMenu::university:

FilterByUniversity(TopMenu::file, numberOfStudents, students);

break;

case BottomMenu::course:

FilterByCourse(TopMenu::file, numberOfStudents, students);

break;

case BottomMenu::group:

FilterByGroup(TopMenu::file, numberOfStudents, students);

break;

case BottomMenu::back:

system("cls");

Fio();

break;

default:

IncorrectOption();

break;

}

}

}

void Fio()

{

cout << "Control work #2\nEfremov Ivan Andreevich\nGroup #423\nVariant #8\n\n";

}

//меню верхнего уровня

void OptionsTop()

{

cout << "1 - Console input\n2 - File input\n3 - Run module tests\n4 - Exit\n\n";

}

void InputOption(int option)

{

if (option == TopMenu::console) {

cout << "\n<<Console Input>>\n\n";

}

else if (option == TopMenu::file) {

cout << "\n<<File Input>>\n\n";

}

}

//меню нижнего уровня

void OptionsBottom()

{

cout << "\n\n1 - List of all students\n";

cout << "2 - Filter students by university\n";

cout << "3 - Filter students by course\n";

cout << "4 - Filter students by group\n";

cout << "5 - Back\n\n";

}

void IncorrectOption()

{

cout << "\nThere is no such option in menu!\n\n";

}

//предложить пользователю сохранить результаты работы программы в файл

char SaveResults(fstream& fout)

{

string name;

cout << "Do you want to save results in the file? (y/n)\n\n";

char res = 'n';

do {

res = GetChar(">>");

if (res != 'y' && res != 'n') {

cout << "Incorrect input. Type 'y' or 'n' only!\n\n";

}

} while (res != 'y' && res != 'n');

if (res == 'y') {

name = OpenFile(WorkWithFiles::output, fout);

}

return res;

}

//открытие файла для чтения или записи

string OpenFile(int option, fstream& file)

{

string name = "";

if (option == WorkWithFiles::input) {

do {

name = GetLink("\nEnter the name of file with data. Example: students.txt\n");

file.open(name, ios::in);

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

cout << "\nError opening file. Make sure, that file exists!\n";

}

} while (!file.is\_open());

return name;

}

else {

do {

name = GetLink("\nEnter the name of file where results will be stored.\nIf there is data in the file it will be overwritten.\nExample: filtered.txt\n");

file.open(name, ios::out, ios::trunc);

} while (!file.is\_open());

return name;

}

}

//основная структура программы

void StartProgram()

{

vector <Student> students;

int actionTop = 1, actionBottom = 1;

Fio();

while (actionTop) {

OptionsTop();

actionTop = GetInt(">>");

InputOption(actionTop);

switch (actionTop)

{

case TopMenu::console:

WorkWithConsole(students, actionBottom);

break;

case TopMenu::file:

WorkWithFile(students, actionBottom);

break;

case TopMenu::module:

LaunchAllTests();

break;

case TopMenu::quit:

cout << "\nProgramm finished it's work!\n";

exit(EXIT\_SUCCESS);

default:

IncorrectOption();

break;

}

}

}

Файл DataInput.h:

#pragma once

#include "UserInterface.h"

void SplitString(string s, string\* ans);

void ConsoleInput(vector<Student> &students, ui & numberOfStudents);

bool FileInput(vector<Student>& students, fstream& fin, ui& numberOfStudents, string name);

Файл DataInput.cpp:

#include "DataInput.h"

#define arguments 6

//разделение строки по пробелам и заполнение массива полученными значениями

void SplitString(string s, string\* ans)

{

string cur = "";

ui cnt = 0;

for (int i = 0; i < arguments; i++) ans[i] = "";

for (int i = 0; i < s.length(); i++) {

if (s[i] == ' ') {

ans[cnt] = cur;

cur = "";

cnt++;

}

else if (i == s.length() - 1) {

cur += s[i];

ans[cnt] = cur;

}

else cur += s[i];

}

}

//ввод данных с консоли

void ConsoleInput(vector<Student>& students, ui& numberOfStudents)

{

string n = "", s = "", a = "", u = "", tmp="";

ui c = 0, g = 0;

numberOfStudents = static\_cast<ui>(GetInt("\nEnter number of students : "));

students.resize(numberOfStudents);

for (ui i = 0; i < numberOfStudents; i++) {

cout << "\nStudent " << i + 1 << ":\n";

n = GetString("Enter name of student #", i + 1);

s = GetString("Enter surname of student #", i + 1);

a = GetString("Enter adress of student #", i + 1);

u = GetString("Enter university of student #", i + 1);

tmp = "Enter course of student #" + to\_string(i + 1) + ": ";

ui g = static\_cast<ui>(GetInt(tmp));

tmp = "Enter group number of student #" + to\_string(i + 1) + ": ";

ui c = static\_cast<ui>(GetInt(tmp));

students[i].SetInformation(n, s, a, u, c, g);

}

//SortData(students);

}

//ввод данных с файла

bool FileInput(vector<Student>& students, fstream &fin, ui &numberOfStudents, string name)

{

string args[arguments];

ui lineNumber = 0, j = 0, course = 0, group = 0;

string tmp = "";

char\* trash;

//подсчет количества данных о разных студентах в файле

while (getline(fin, tmp)) {

if (!tmp.empty()) numberOfStudents++;

}

fin.close();

if (numberOfStudents == 0) {

cout << "ERROR!\nThere is no correct data in the file\n";

cout << "Choose another file or edit the current one\n\n";

return false;

}

students.resize(numberOfStudents);

fin.open(name);

while (getline(fin, tmp) && j < numberOfStudents) {

lineNumber++;

if (!tmp.empty()) {

SplitString(tmp, args);

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

if (args[i] == "" || args[i] == " ") {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nIt may be incorrect space input or incorrect amount of values\n";

cout << "Choose another file or edit the current one\n\n";

students.clear();

return false;

}

}

course = (strtol(args[4].c\_str(), &trash, 10));

group = (strtol(args[5].c\_str(), &trash, 10));

//проверка на положительность значения курса

if (course <= 0) {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nCourse number must be >0!\n";

cout << "Choose another file or edit the current one\n\n";

students.clear();

return false;

}

//проверка на положительность значения группы

if (group <= 0) {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nGroup number must be >0!\n";

cout << "Choose another file or edit the current one\n\n";

students.clear();

return false;

}

//проверка перевелась ли строка полностью к типу unsigned int

if (static\_cast<ui>(course) != static\_cast<ui>(stoi(args[4]))) {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nCourse number must be integer!\n";

cout << "Choose another file or edit the current one\n\n";

students.clear();

return false;

}

if (static\_cast<ui>(group) != static\_cast<ui>(stoi(args[5]))) {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nGroup number must be integer!\n";

cout << "Choose another file or edit the current one\n\n";

students.clear();

return false;

}

//заполнение полей класса для i-того элемента

students[j].SetInformation(args[0], args[1], args[2], args[3], course, group);

j++;

}

}

fin.close();

return true;

}

Файл GetInfo.h:

#pragma once

#include <iostream>

#include <limits>

#include <string>

using namespace std;

int GetInt(string s);

string GetString(string s, int i);

string GetLink(string s);

string GetString(string s);

char GetChar(string s);

Файл GetInfo.cpp:

#include "GetInfo.h"

//очистка потока ввода

void IgnoreLine()

{

cin.ignore(numeric\_limits<std::streamsize>::max(), '\n');

}

//считывание целого числа с консоли

int GetInt(string s) {

cout << s;

int tmp = 0;

while (true) {

cin >> tmp;

if (cin.fail()) {

cin.clear();

IgnoreLine();

cout << "\nUse correct values!\n\n";

cout << s;

}

else if (tmp <= 0) {

cout << "\nValue must be >0\n\n";

cin.clear();

IgnoreLine();

cout << s;

}

else {

IgnoreLine();

return tmp;

}

}

}

//считывание строки с консоли, указывая номер объекта для которого строка считывается

string GetString(string s, int i)

{

string a = "";

while (true) {

cout << s << i << ": ";

cin >> a;

if (getchar() != '\n') {

cout << "\nInput only one word per line!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else if (a.empty()) {

cout << "\nYou must enter a word!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else return a;

}

}

//считывание всей строки для указания адреса файла

string GetLink(string s)

{

string b = "";

cout << s;

cout << "Path must not contain cyrillic symbols!\n\n>>";

getline(cin, b);

return b;

}

//считывание строки без нумерации

string GetString(string s)

{

string b = "";

while (true) {

cout << s;

cin >> b;

if (getchar() != '\n') {

cout << "\nInput only one word per line!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else if (b.empty()) {

cout << "\nYou must enter a word!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else return b;

}

return b;

}

//считывание ответа y/n

char GetChar(string s)

{

char tmp = '\n';

cout << s;

while (true) {

tmp = getchar();

if (tmp == '\n') {

cout << "You must type y or n!\n\n";

cout << s;

cin.clear();

IgnoreLine();

}

else if (getchar() != '\n') {

cout << "Incorrect input! Type only y or n!\n\n";

cout << s;

cin.clear();

IgnoreLine();

}

else return tmp;

}

}

Файл Tasks.h:

#pragma once

#include "UserInterface.h"

bool Comparator(Student x, Student y);

void Sort(vector<Student>& students);

bool Comparison(string university, string chosenUniversity);

void PrintAll(int mode, ui numberOfStudents, vector<Student>& students);

void FilterByUniversity(int mode, ui numberOfStudents, vector<Student>& students);

void FilterByCourse(int mode, ui numberOfStudents, vector<Student>& students);

void FilterByGroup(int mode, ui numberOfStudents, vector<Student>& students);

void Print(int mode, vector<Student>& filteredStudents, string line);

Файл Tasks.cpp:

#include "Tasks.h"

#include <algorithm>

//компаратор, задающий условия сортировки

bool Comparator(Student x, Student y)

{

if (x.GetUniversity() != y.GetUniversity()) return x.GetUniversity() < y.GetUniversity();

if (x.GetName() != y.GetName()) return x.GetName() < y.GetName();

if (x.GetSurname() != y.GetSurname()) return x.GetSurname() < y.GetSurname();

if (x.GetAdress() != y.GetAdress()) return x.GetAdress() < y.GetAdress();

if (x.GetCourse() != y.GetCourse()) return x.GetCourse() < y.GetCourse();

return x.GetGroup() < y.GetGroup();

}

//сортировка вектора

void Sort(vector<Student>& students)

{

sort(students.begin(), students.end(), Comparator);

}

//сравнение строк без учета регистра

bool Comparison(string university, string chosenUniversity)

{

if (university == chosenUniversity) return true;

else {

if (university.length() == chosenUniversity.length()) {

for (int i = 0; i < university.length(); i++) {

if (tolower(university[i]) != tolower(chosenUniversity[i])) {

return false;

}

}

return true;

}

else return false;

}

}

//фильтр студентов по университету

void FilterByUniversity(int mode, ui numberOfStudents, vector<Student> &students)

{

vector<Student> filtered;

string chosenUniversity = GetString("\nEnter the university you want to see students from: ");

for (ui i = 0; i < numberOfStudents; i++) {

if (Comparison(students[i].GetUniversity(), chosenUniversity)) {

filtered.push\_back(students[i]);

}

}

string line = "Students from the " + chosenUniversity + " university:\n\n";

Print(mode, filtered, line);

}

//фильтр студентов по курсу

void FilterByCourse(int mode, ui numberOfStudents, vector<Student>& students)

{

vector<Student> filtered;

ui chosenCourse = static\_cast<ui>(GetInt("\nEnter the course you want to see students from: "));

for (ui i = 0; i < numberOfStudents; i++) {

if (students[i].GetCourse()==chosenCourse) {

filtered.push\_back(students[i]);

}

}

string line = "Students from the " + to\_string(chosenCourse) + " course:\n\n";

Print(mode, filtered, line);

}

//фильтр студентов по группе

void FilterByGroup(int mode, ui numberOfStudents, vector<Student>& students)

{

vector<Student> filtered;

ui chosenGroup = static\_cast<ui>(GetInt("\nEnter the group you want to see students from: "));

for (ui i = 0; i < numberOfStudents; i++) {

if (students[i].GetGroup() == chosenGroup) {

filtered.push\_back(students[i]);

}

}

string line = "Students from the " + to\_string(chosenGroup) + " group:\n\n";

Print(mode, filtered, line);

}

//вывод на консоль и в файл

void Print(int mode, vector<Student>& filteredStudents, string line)

{

fstream fout;

sort(filteredStudents.begin(), filteredStudents.end(), Comparator);

if (mode == TopMenu::console) {

if (!filteredStudents.size()) {

cout << "There is no students by such category!\n\n";

return;

}

sort(filteredStudents.begin(), filteredStudents.end(), Comparator);

cout << line;

for (ui i = 0; i < filteredStudents.size(); i++) {

filteredStudents[i].ShowInformation(i + 1);

}

char ans = SaveResults(fout);

if (ans == 'y') {

fout << line;

for (ui i = 0; i < filteredStudents.size(); i++) {

fout << filteredStudents[i].GetAll(i + 1);

}

fout.close();

}

}

else {

if (!filteredStudents.size()) {

cout << "\nThere is no students by such category!\nData hasn't been written in the file!\n";

return;

}

string name = OpenFile(WorkWithFiles::output, fout);

fout << line;

for (ui i = 0; i < filteredStudents.size(); i++) {

fout << filteredStudents[i].GetAll(i + 1);

}

fout.close();

}

}

Файл Person.h:

#pragma once

#include "GetInfo.h"

class Person{

protected:

string surname;

string adress;

public:

Person();

void SetInformation(string s, string a);

string GetSurname();

string GetAdress();

};

Файл Person.cpp:

#include "Person.h"

//конструктор по умолчанию

Person::Person()

{

surname = "";

adress = "";

}

//заполнение полей класса

void Person::SetInformation(string s, string a)

{

surname = s;

adress = a;

}

//получить поле "фамилия"

string Person::GetSurname()

{

return surname;

}

//получить поле "адрес"

string Person::GetAdress()

{

return adress;

}

Файл Student.h:

#pragma once

#include "Person.h"

#define ui unsigned int

class Student : public Person{

private:

string name;

string university;

ui course;

ui group;

public:

Student();

void SetInformation(string n, string s, string a, string u, ui c, ui g);

void ShowInformation(ui number);

string GetAll(ui number);

string GetName();

string GetUniversity();

ui GetCourse();

ui GetGroup();

};

Файл Student.cpp:

#include "Student.h"

//конструктор по умолчанию

Student::Student()

{

name = "";

surname = "";

adress = "";

university = "";

course = 1;

group = 1;

}

//заполнение полей класса

void Student::SetInformation(string n, string s, string a, string u, ui c, ui g)

{

name = n;

surname = s;

adress = a;

university = u;

course = c;

group = g;

}

//вывод информации об объекте на консоль

void Student::ShowInformation(ui number)

{

cout << "Information about student #" << number << ":\n";

cout << "Name: " << name << endl;

cout << "Surname: " << surname << endl;

cout << "Adress: " << adress << endl;

cout << "University: " << university << endl;

cout << "Course: " << course << endl;

cout << "Group: " << group << endl << endl;

}

//получить информацию об объекте в виде строки

string Student::GetAll(ui number)

{

return "Information about student #" + to\_string(number) + ":\n"

+ "Name: " + name + '\n' + "Surname: " + surname + '\n'

+ "Adress: " + adress + '\n' + "University: " + university + '\n'

+ "Course: " + to\_string(course) + '\n' + "Group: " + to\_string(group) + "\n\n";

}

//получить поле "имя"

string Student::GetName()

{

return name;

}

//получить поле "университет"

string Student::GetUniversity()

{

return university;

}

//получить поле "курс"

ui Student::GetCourse()

{

return course;

}

//получить поле "группа"

ui Student::GetGroup()

{

return group;

}

Файл ModuleTests.h:

#pragma once

#include "Student.h"

#include "Tasks.h"

enum ValuesForModuleTests { first = 2, second = 2, third = 3, fourth = 7 };

bool Check(vector<Student> &test, vector<Student> &fileTest, int lim);

bool Test1();

bool Test2();

bool Test3();

bool Test4();

bool Test5();

void LaunchAllTests();

Файл ModuleTests.cpp:

#include "ModuleTests.h"

#include "DataInput.h"

#include <sstream>

bool Check(vector<Student>& testArray, vector<Student>& testArrayFile, int lim)

{

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

if (testArray[i].GetName() != testArrayFile[i].GetName() ||

testArray[i].GetSurname() != testArrayFile[i].GetSurname() ||

testArray[i].GetAdress() != testArrayFile[i].GetAdress() ||

testArray[i].GetUniversity() != testArrayFile[i].GetUniversity() ||

testArray[i].GetCourse() != testArrayFile[i].GetCourse() ||

testArray[i].GetGroup() != testArrayFile[i].GetGroup()

) return false;

}

return true;

}

bool Test1()

{

//проверка ввода данных с файла

cout << "\tTEST#1\n\tTesting file input...\n\n";

fstream fin;

ui numberOfStudents = 0;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

vector<Student> test\_array(ValuesForModuleTests::first);

vector<Student> test\_array\_file;

test\_array[0].SetInformation("Ilya", "Ekkert", "Berlin", "ITMO", 1, 978);

test\_array[1].SetInformation("Dmitriy", "Kaplunov", "Krasnoyarsk", "SIBGU", 2, 576);

if (!FileInput(test\_array\_file, fin, numberOfStudents, fileName)) {

cout << "FAILURE!\nValues from inside the program and values from the test file do not match!\n\n";

return false;

}

if (!Check(test\_array, test\_array\_file, ValuesForModuleTests::first)) {

cout << "FAILURE!\nValues from inside the program and values from the test file do not match!\n\n";

return false;

}

return true;

}

bool Test2()

{

//фильтр по университету

cout << "\tTEST#2\n\tTesting unviersity filter function...\n\n";

fstream fin;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

string res = "", resFile = "";

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

ui numberOfStudents = 0;

vector<Student> test\_array(ValuesForModuleTests::second);

test\_array[0].SetInformation("Ilya", "Ekkert", "Berlin", "ITMO", 1, 978);

test\_array[1].SetInformation("Dmitriy", "Kaplunov", "Krasnoyarsk", "SIBGU", 2, 576);

res = test\_array[1].GetUniversity();

vector<Student> test\_array\_file;

if (!FileInput(test\_array\_file, fin, numberOfStudents, fileName)) {

cout << "FAILURE!\nValues from inside the program and values from the test file do not match!\n\n";

return false;

}

for (ui i = 0; i < numberOfStudents; i++) {

if (Comparison(test\_array\_file[i].GetUniversity(), res)) {

resFile = test\_array\_file[i].GetUniversity(); break;

}

}

if (res != resFile) {

cout << "FAILURE!\nIncorrect answer\n\n";

return false;

}

return true;

}

bool Test3()

{

//фильтр по группе

cout << "\tTEST#3\n\tTesting group filter function...\n\n";

fstream fin;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

ui res = 0, resFile = 0;

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

ui numberOfStudents = 0;

vector<Student>test\_array(ValuesForModuleTests::third);

test\_array[0].SetInformation("Ilya", "Ekkert", "Berlin", "ITMO", 1, 978);

test\_array[1].SetInformation("Dmitriy", "Kaplunov", "Krasnoyarsk", "SIBGU", 2, 576);

test\_array[2].SetInformation("Mikhail", "Zubenko", "Moldavia", "MGU", 1, 123);

vector<Student>test\_array\_file;

if (!FileInput(test\_array\_file, fin, numberOfStudents, fileName)) {

cout << "FAILURE!\nValues from inside the program and values from the test file do not match!\n\n";

return false;

}

res = test\_array[2].GetGroup();

for (ui i = 0; i < numberOfStudents; i++) {

if (test\_array\_file[i].GetGroup() == res) {

resFile = test\_array\_file[i].GetGroup(); break;

}

}

if (resFile != res) {

cout << "FAILURE!\nIncorrect answer\n\n";

return false;

}

return true;

}

bool Test4()

{

//корректность счёта строк в файле

cout << "\tTEST#4\n\tTesting correctness of counting lines in a file...\n\n";

fstream fin;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

ui numberOfStudents = 0;

vector<Student> test\_array(ValuesForModuleTests::fourth);

if (!FileInput(test\_array, fin, numberOfStudents, fileName)) {

cout << "FAILURE!\nValues from inside the program and values from the test file do not match!\n\n";

return false;

}

if (numberOfStudents != ValuesForModuleTests::fourth) {

cout << "FAILURE!\nIncorrect amount of data in the file!\n\n";

return false;

}

return true;

}

bool Test5(){

//корректность работы ввода с консоли

cout << "\tTEST#5\n\tTesting console input...\n\n";

{

istringstream test("576\n\n");

streambuf\* cinbuf = cin.rdbuf(test.rdbuf());

bool res = (GetInt("") == 576);

if (!res) {

cout << "FAILURE!\nIncorrect console input!\n\n";

return false;

}

cin.rdbuf(cinbuf);

}

return true;

}

void LaunchAllTests()

{

bool tests[5] = { Test1(), Test2(), Test3(), Test4(), Test5() };

for (ui i = 0; i < 5; i++) {

if (!tests[i]) cout << "TEST#" << i + 1 << "\tFAILURE!\n\n";

}

if (tests[0] && tests[1] && tests[2] && tests[3] && tests[4]) cout << "All tests SUCCEDED!\n\n";

else cout << "Some of the tests failed\n\n";

}