Московский авиационный институт

(Национальный исследовательский университет)

Факультет №3:

*Системы управления, информатика и электроэнергетика*.

Кафедра 304.

Пояснительная записка к курсовой работе

По учебной дисциплине «Программирование, структуры и алгоритмы обработки данных»

Группа: М3О-225Бк-21

Вариант №9

Выполнил:

*Новиков К.А.*

Приняла:

*Дмитриевна Е.А.*

Москва 2022 г.

Оглавление

[Постановка задачи: 4](#_Toc123131029)

[Процесс разработки 5](#_Toc123131030)

[Элемент списка 5](#_Toc123131031)

[Класс Node 5](#_Toc123131032)

[Класс StudentNode 6](#_Toc123131033)

[Список 7](#_Toc123131034)

[Блок-схемы некоторых алгоритмов 8](#_Toc123131035)

[Добавление узла «Студент» 8](#_Toc123131036)

[Печать группы 9](#_Toc123131037)

[Код программы 10](#_Toc123131038)

[Файл «CW1 3s-1c.cpp» 10](#_Toc123131039)

[Файл «Menu.h» 11](#_Toc123131040)

[Файл «Menu.cpp» 11](#_Toc123131041)

[Файл «Struct.h» 16](#_Toc123131042)

[Файл «Struct.cpp» 19](#_Toc123131043)

[Файл «Line.h» 44](#_Toc123131044)

[Файл «Line.cpp» 47](#_Toc123131045)

[Файл «Console.h» 74](#_Toc123131046)

[Файл «Console.cpp» 74](#_Toc123131047)

[Файл «Char.h» 79](#_Toc123131048)

[Файл «Char.cpp» 79](#_Toc123131049)

[Файл «Error.h» 83](#_Toc123131050)

[Файл «Error.cpp» 83](#_Toc123131051)

[Файл «Functions.h» 87](#_Toc123131052)

[Файл «Functions.cpp» 87](#_Toc123131053)

[Результат работы программы 114](#_Toc123131054)

[Запуск и хранение данных 114](#_Toc123131055)

[Меню 115](#_Toc123131056)

[1 Загрузить из файла 115](#_Toc123131057)

[2 Выгрузить в файл 115](#_Toc123131058)

[3 Удалить всё 116](#_Toc123131059)

[4 Взаимодействие со студентами 116](#_Toc123131060)

[5 Взаимодействие с ведомостями 118](#_Toc123131061)

[6 Взаимодействие с группами 121](#_Toc123131062)

[7 Список не загруженных линий 121](#_Toc123131063)

[Вывод: 122](#_Toc123131064)

# **Постановка задачи**:

Разработать систему управления базой данных на языке C++:

Система учёта студентов: фамилия, имя, отчество, номер группы, пол, дата рождения (год, месяц, день), домашний адрес, год поступления в вуз, результаты обучения по сессиям.

Для хранения данных использовать текстовый файл. Разработать структуры данных для хранения данных и функции для их обработки. Разработать функции для ввода данных, выборки значения по заданному условию, вывода данных на экран, удаления, записи новых данных. Возможно наличие нескольких узлов списков с одинаковыми значениями полей. Реализовать систему меню.

# Процесс разработки

Для хранения элементов БД в программе была выбрана структура «Двунаправленный список». Каждый элемент такой структуры имеет в себе поля с нужными данными, а так же указатель на следующий и предыдущий элемент. Для работы с таким списком в программе хранятся 2 указателя: на начало и конец данного списка. При удалении/добавлении элемента списка меняются связи между узлами списка.

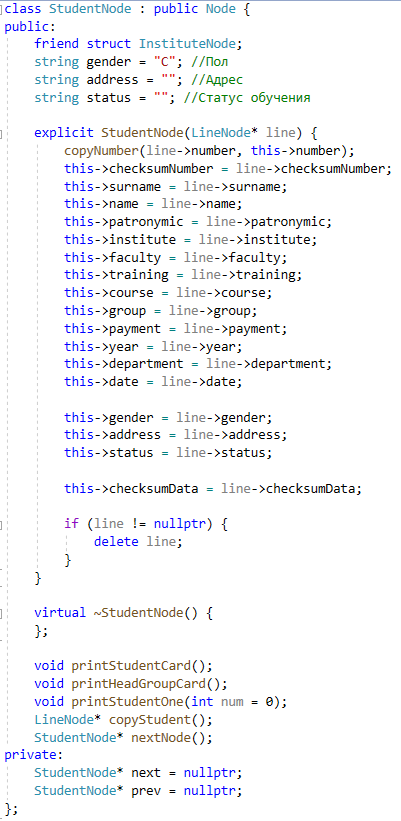
## Элемент списка

В программе есть 2 основных и 1 побочных элемента списка. Первые 2 – это классы StudentNode и RecordNode (для студентов и ведомостей), которые содержат в себе поля с данными и некоторые функции их обработки. Ссылки на следующий и предыдущий элемент являются приватными полями, с целью отсутствия возможности нарушений связи между элементами списка, но для их получения (не дружественной структурой) предусмотрены необходимые фукнции. Оба класса имеют много одинаковых полей, поэтому с целью предотвращения дублирования основные поля наследуются от класса Node.

### Класс Node

****

### Класс StudentNode



Классы StudentNode и RecordNode содержат в себе конструктор от класса LineNode – третий элемент списка. В данный класс осуществляется считывание, запись и последующая обработка (на корректность) данных для последующей передачи их данным 2-м классам.

## Список

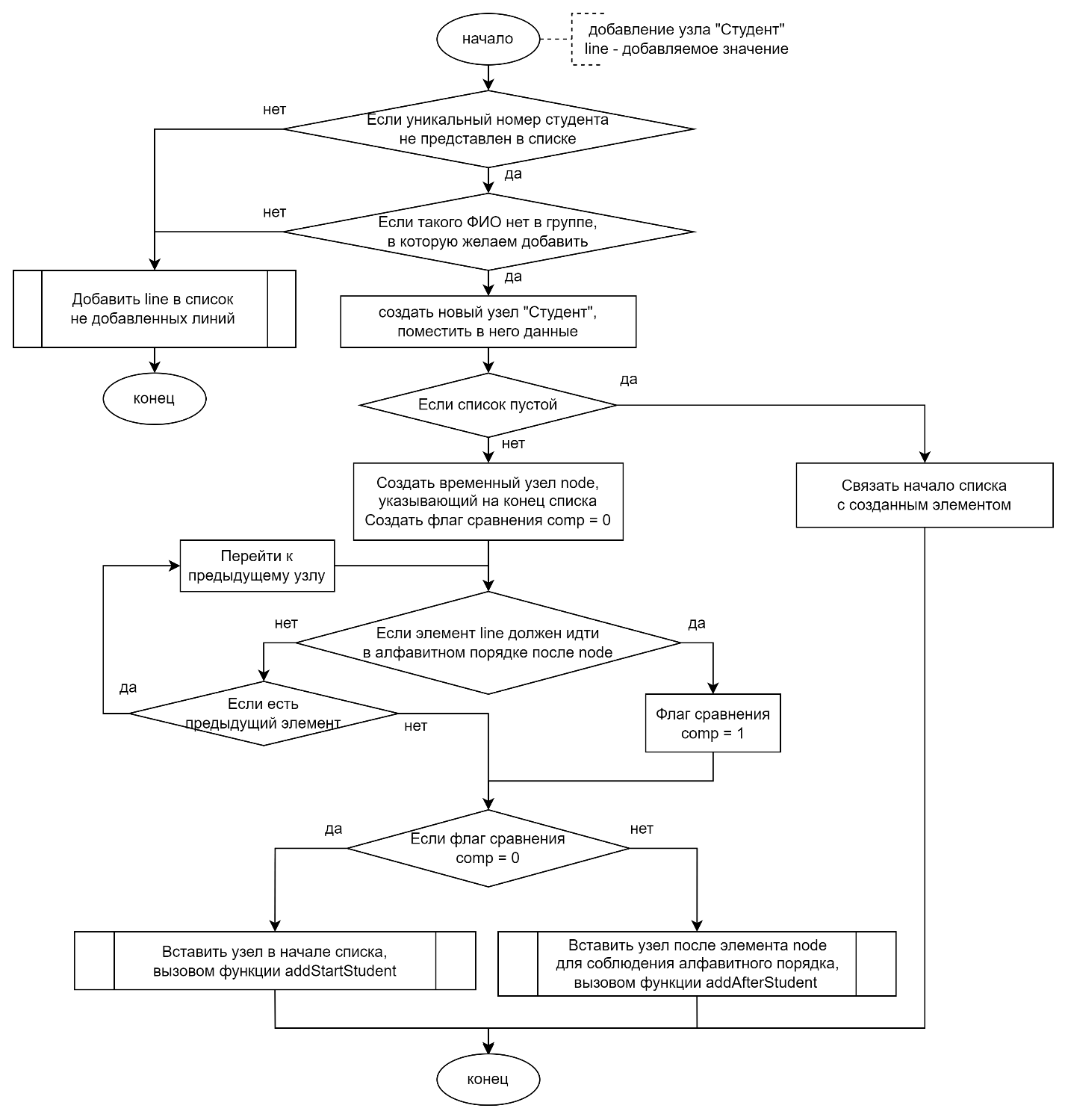
Список представляет собой класс InstituteNode, который содержит указатель на первый и последний элемент списков записей «Студент» и «Ведомость», а также указатель на первый и последний элемент однонаправленного списка состоящего из элементов LineNode. Данный список хранит в себе неудачно обработанные записи из файла.

Класс InstituteNode имеет следующие основные функции для работы со списком:

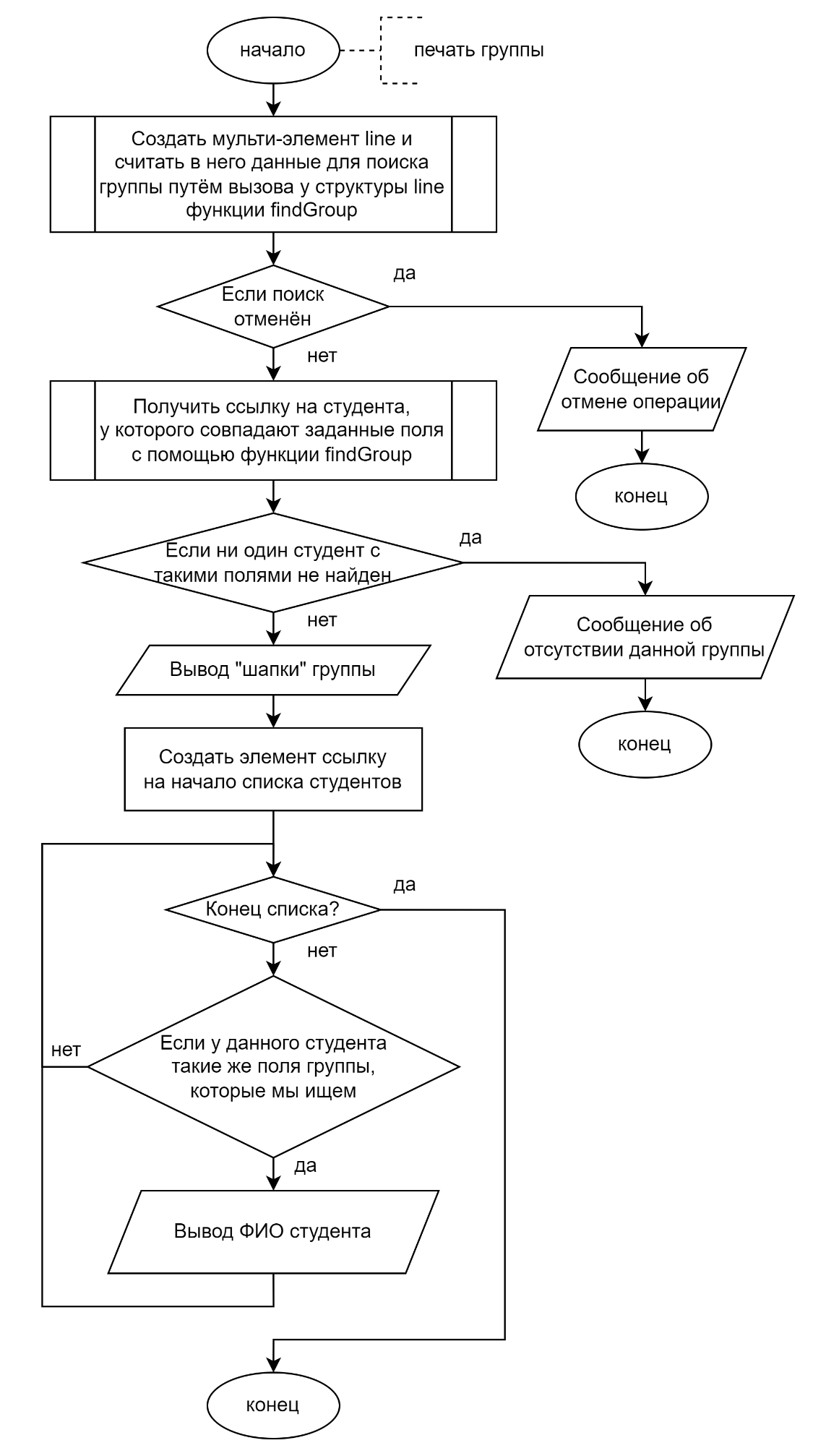
* **вставка** нового элемента - add. Функция принимает элемент LineNode, проводит дополнительные проверки перед вставкой, а затем вставляет в список rkfccjd StudentNode, RecordNode или LineNode, в зависимости от добавляемых данных. В первые 2 списка элементы вставляются в алфавитном порядке, с использованием приватных функций addStartStudent, addAfterStudent, addStartRecord, addAfterRecord.
* **удаление** элемента списка - deleteOneStydent, deleteOneRecord. Функции удаляют один элемент из списка, подходящий по заданным параметрам.
* **удаление** нескольких элементов списка – deleteRecord, deleteGroup. Функции удаляют все элементы списка, подходящие по заданным параметрам.
* **очистка** списков – deleteAll. Функция полностью очищает 3 списка.

## Блок-схемы некоторых алгоритмов

### Добавление узла «Студент»



### Печать группы



# Код программы

### Файл «CW1 3s-1c.cpp»

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* КАФЕДРА № 304 2 КУРС \*

\*---------------------------------------------------------------\*

\* Project Type : Win32 Console Application \*

\* Project Name : CW1 3s-1c \*

\* File Name : CW1 3s-1c.cpp \*

\* Language : C/C++ \*

\* Programmer(s) : Новиков К.А. \*

\* Modifyed By : \*

\* Lit source : \*

\* Created : 16/12/2022 \*

\* Last Revision : 27/12/2022 \*

\* Comment(s) : Курсовая работа списки. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

#include "Struct.h"

#include "Menu.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "RU");

SetConsoleOutputCP(65001);

SetConsoleCP(65001);

srand(time(nullptr));

SetConsoleTitle(L"БД института");

consoleCursorVisible(false);

cout << "Начало работы программы" << endl;

//test();

InstituteNode institute = createInstitutesList();

menuStop();

char key;

bool run = 1;

system("CLS");

printCommands();

while (run) {

consoleCursorVisible(false);

key = \_getch();

//cout << "Код:" << (int)key << endl;

switch (key) {

case kESC:

run = 0;

break;

case k1:

menuReadFile(institute);

break;

case k2:

menuWriteFile(institute);

break;

case k3:

menuDeleteAll(institute);

break;

case k4:

menuStudents(institute);

break;

case k5:

menuRecords(institute);

break;

case k6:

menuGroups(institute);

break;

case k7:

menuFails( institute);

break;

default:

break;

}

}

}

### Файл «Menu.h»

#ifndef CW\_MENU\_H

#define CW\_MENU\_H

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

#include "Struct.h"

#include "Char.h"

#include "Functions.h"

#include "Console.h"

using namespace std;

void menuStop();

void printCommands();

void menuReadFile(InstituteNode& institute);

void menuWriteFile(InstituteNode& institute);

void menuDeleteAll(InstituteNode& institute);

void menuStudents(InstituteNode& institute);

void menuRecords(InstituteNode& institute);

void menuGroups(InstituteNode& institute);

void menuFails(InstituteNode& institute);

#endif

### Файл «Menu.cpp»

#include "Menu.h"

using namespace std;

void menuStop() {

cout << endl << "Для взаимодействия с программой нажмите ENTER";

char key;

while (true) {

key = \_getch();

switch (key) {

case kENTER:

return;

default:

break;

}

}

}

void printCommands() {

cout << "Нажмите соотвествующую клавишу:" << endl;

cout << "1 Загрузить из файла" << endl;

cout << "2 Выгрузить в файл" << endl;

cout << "3 Удалить всё" << endl;

cout << "4 Взаимодействие со студентами" << endl;

cout << "5 Взаимодействие с ведомостями" << endl;

cout << "6 Взаимодействие с группами" << endl;

cout << "7 Список незагруженных линий" << endl;

cout << "ESC остановить программу" << endl;

}

void menuReadFile(InstituteNode& institute) {

consoleCursorVisible(false);

system("CLS");

cout << "Введите название файла для загрузки" << endl;

consoleCursorVisible(true);

getline(cin, institute.fileToWork);

consoleCursorVisible(false);

readFromFile(institute);

menuStop();

system("CLS");

printCommands();

}

void menuWriteFile(InstituteNode& institute) {

consoleCursorVisible(false);

system("CLS");

cout << "Введите название файла для выгрузки" << endl;

consoleCursorVisible(true);

getline(cin, institute.fileToWork);

consoleCursorVisible(false);

cout << "Сохранить не загрузившиеся записи?" << endl;

cout << "1 Да" << endl;

cout << "2 Нет" << endl;

cout << "ESC назад в главное меню" << endl;

char key;

bool run = 1;

while (run) {

key = \_getch();

switch (key) {

case kESC:

system("CLS");

printCommands();

return;

case k1:

writeToFile(institute, 1);

run = 0;

break;

case k2:

writeToFile(institute, 0);

run = 0;

break;

default:

break;

}

}

menuStop();

system("CLS");

printCommands();

}

void menuDeleteAll(InstituteNode& institute) {

consoleCursorVisible(false);

system("CLS");

cout << "Действительно всё удалить?" << endl;

cout << "1 Да" << endl;

cout << "2 Нет" << endl;

cout << "ESC назад в главное меню" << endl;

char key;

bool run = 1;

while (run) {

key = \_getch();

switch (key) {

case kESC:

case k2:

run = 0;

break;

case kENTER:

case k1:

institute.deleteAll();

run = 0;

break;

default:

break;

}

}

menuStop();

system("CLS");

printCommands();

}

void menuStudents(InstituteNode& institute) {

consoleCursorVisible(false);

system("CLS");

cout << "Нажмите соответсвующую клавишу:" << endl;

cout << "1 Создать нового студента" << endl;

cout << "2 Редактировать студента" << endl;

cout << "3 Удалить студента" << endl;

cout << "4 Вывести студента" << endl;

cout << "5 Cписок записей студентов" << endl;

cout << "ESC назад в главное меню" << endl;

char key;

bool run = 1;

while (run) {

key = \_getch();

switch (key) {

case kESC:

run = 0;

break;

case k1:

{

createStydent(institute);

run = 0;

break;

}

case k2:

editStydent(institute);

run = 0;

break;

case k3:

deleteStydent(institute);

run = 0;

break;

case k4:

printStydent(institute);

run = 0;

break;

case k5:

institute.printLineStudents();

run = 0;

break;

default:

break;

}

}

menuStop();

system("CLS");

printCommands();

}

void menuRecords(InstituteNode& institute) {

consoleCursorVisible(false);

system("CLS");

cout << "Нажмите соответсвующую клавишу:" << endl;

cout << "1 Взять новую ведомость" << endl;

cout << "2 Редактировать ведомость" << endl;

cout << "3 Удалить ведомость" << endl;

cout << "4 Вывести ведомость" << endl;

cout << "5 Cписок записей ведомостей" << endl;

cout << "ESC назад в главное меню" << endl;

char key;

bool run = 1;

while (run) {

key = \_getch();

switch (key) {

case kESC:

run = 0;

break;

case k1:

{

createRecord(institute);

run = 0;

break;

}

case k2:

editRecord(institute);

run = 0;

break;

case k3:

deleteRecord(institute);

run = 0;

break;

case k4:

printRecord(institute);

run = 0;

break;

case k5:

institute.printLineRecords();

run = 0;

break;

default:

break;

}

}

menuStop();

system("CLS");

printCommands();

}

void menuGroups(InstituteNode& institute) {

consoleCursorVisible(false);

system("CLS");

cout << "Нажмите соответсвующую клавишу:" << endl;

cout << "1 Удалить группу" << endl;

cout << "2 Вывести группу" << endl;

cout << "ESC назад в главное меню" << endl;

char key;

bool run = 1;

while (run) {

key = \_getch();

switch (key) {

case kESC:

run = 0;

break;

case k1:

{

deleteGroup(institute);

run = 0;

break;

}

case k2:

printGroup(institute);

run = 0;

break;

default:

break;

}

}

menuStop();

system("CLS");

printCommands();

}

void menuFails(InstituteNode& institute) {

system("CLS");

if (institute.failList != nullptr) {

cout << "Список незагруженных линий и ошибки в них:" << endl;

institute.printFailList(1);

}

else {

cout << "Не загруженные линии отсутствуют" << endl;

}

menuStop();

system("CLS");

printCommands();

}

### Файл «Struct.h»

#ifndef CW\_STRUCT\_H

#define CW\_STRUCT\_H

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

#include "Char.h"

#include "Error.h"

#include "Console.h"

#include "Line.h"

using namespace std;

struct InstituteNode; //Институт

struct StudentNode; //Студент

struct RecordNode; //Ведомость

class StudentNode : public Node {

public:

friend struct InstituteNode;

string gender = "С"; //Пол

string address = ""; //Адрес

string status = ""; //Статус обучения

explicit StudentNode(LineNode\* line) {

copyNumber(line->number, this->number);

this->checksumNumber = line->checksumNumber;

this->surname = line->surname;

this->name = line->name;

this->patronymic = line->patronymic;

this->institute = line->institute;

this->faculty = line->faculty;

this->training = line->training;

this->course = line->course;

this->group = line->group;

this->payment = line->payment;

this->year = line->year;

this->department = line->department;

this->date = line->date;

this->gender = line->gender;

this->address = line->address;

this->status = line->status;

this->checksumData = line->checksumData;

if (line != nullptr) {

delete line;

}

}

virtual ~StudentNode() {

};

void printStudentCard();

void printHeadGroupCard();

void printStudentOne(int num = 0);

LineNode\* copyStudent();

StudentNode\* nextNode();

private:

StudentNode\* next = nullptr;

StudentNode\* prev = nullptr;

};

class RecordNode : public Node {

public:

friend struct InstituteNode;

string subject = ""; //Предмет

string teacherSurname = ""; //Фамилия учителя

string teacherName = ""; //Имя учителя

string teacherPatronymic = ""; //Отчество учителя

string value = ""; //Оценка

string room = ""; //Уникальный номер ведомости

explicit RecordNode(LineNode\* line) {

copyNumber(line->number, this->number);

this->checksumNumber = line->checksumNumber;

this->surname = line->surname;

this->name = line->name;

this->patronymic = line->patronymic;

this->institute = line->institute;

this->faculty = line->faculty;

this->training = line->training;

this->course = line->course;

this->group = line->group;

this->payment = line->payment;

this->year = line->year;

this->department = line->department;

this->date = line->date;

this->subject = line->subject;

this->teacherSurname = line->teacherSurname;

this->teacherName = line->teacherName;

this->teacherPatronymic = line->teacherPatronymic;

this->value = line->value;

this->room = line->room;

this->checksumData = line->checksumData;

if (line != nullptr) {

delete line;

}

}

~RecordNode() {

};

int editRecord(LineNode\* line, int all, int now);

void printRoom();

void printHeadRecordCard();

void printHeadRecordCardSNP();

void printStudentOne();

void printEditRecordOne(int num);

void printRecordOne(int num);

LineNode\* copyRecord();

RecordNode\* nextNode();

private:

RecordNode\* next = nullptr;

RecordNode\* prev = nullptr;

};

class InstituteNode {

public:

//string path = "Data\\";

string path = "";

//string file = "InstituteData";

string file = "data";

string expansion = ".txt";

string fileToWork = "";

string fileTheName = "";

int nodeSuccess = 0;

int nodeMistake = 0;

int operationSuccess = 0;

int operationMistake = 0;

string\* worlEnd\_запись = new string[] {"записей", "запись", "записи"};

string\* worlEnd\_линия = new string[] { "линий", "линия", "линии" };

explicit InstituteNode() {

}

~InstituteNode() {

if (worlEnd\_запись == nullptr) {

delete[] worlEnd\_запись;

}

if (worlEnd\_линия == nullptr) {

delete[] worlEnd\_линия;

}

};

StudentNode\* headStudentsList = nullptr;

StudentNode\* endStudentsList = nullptr;

RecordNode\* headRecordsList = nullptr;

RecordNode\* endRecordsList = nullptr;

LineNode\* failList = nullptr;

LineNode\* lastFailList = nullptr;

void resetSM();

int checkUniqueSNP(LineNode\* line);

int compareSNP(StudentNode\* node, LineNode\* line);

int checkUniqueGroup(LineNode\* line);

int compareGroup(StudentNode\* node, LineNode\* line);

int checkUniqueNumber(string\* number);

int compareNumber(string\* str1, string\* str2);

int checkUniqueRoom(LineNode\* line);

int compareRoom(RecordNode\* node, LineNode\* line);

string generateRecordRoom();

void generateNumber(LineNode\* line);

StudentNode\* findStudent(LineNode\* line);

RecordNode\* findRecord(LineNode\* line);

StudentNode\* findGroup(LineNode\* line);

void add(LineNode\* line);

void deleteAll();

void deleteOneStydent(string\* number);

void deleteOneStydent(StudentNode\* node);

void deleteOneRecord(RecordNode\* node);

void deleteRecord(LineNode\* node);

void deleteGroup(LineNode\* line);

void printLineStudents();

void printLineRecords();

void printFailList(int num = 0);

private:

void addStudent(LineNode\* line);

void addRecord(LineNode\* line);

void addFail(LineNode\* line);

void addStartStudent(StudentNode\* newStydent);

void addAfterStudent(StudentNode\* node, StudentNode\* newStydent);

void addStartRecord(RecordNode\* newRecord);

void addAfterRecord(RecordNode\* node, RecordNode\* newRecord);

};

#endif

### Файл «Struct.cpp»

#include "Struct.h"

using namespace std;

void StudentNode::printStudentCard() {

system("CLS");

int size = 8;

string arrFillIn[] = { "Студент", "Фамилия", "Имя", "Отчество", "Дата рождения", "Пол", "Адрес", "Статус обучения" };

consoleCoutLetter("|", 0, size);

consoleCoutLetter("|", 47, size);

consoleCoutMesage(arrFillIn, 2, size);

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

consoleCursorGoToXY(19, i);

if (i == 0) {

cout << getGroup();

}

if (i == 1) {

cout << surname;

}

if (i == 2) {

cout << name;

}

if (i == 3) {

cout << patronymic;

}

if (i == 4) {

cout << date;

}

if (i == 5) {

cout << gender;

}

if (i == 6) {

cout << address;

}

if (i == 7) {

cout << getStatus(status);

}

}

cout << endl;

}

void StudentNode::printHeadGroupCard() {

system("CLS");

consoleCoutLetter("|", 0, 1);

consoleCoutLetter("|", 82, 1);

consoleCursorGoToXY(2, 0);

cout << "Группа";

consoleCursorGoToXY(19, 0);

cout << getGroup();

consoleCursorGoToXY(0, 1);

}

void StudentNode::printStudentOne(int num) {

consoleCoutLetter("|", 0, 1, num);

consoleCoutLetter("|", 82, 1, num);

consoleCursorGoToXY(2, 0 + num);

cout << num;

consoleCursorGoToXY(4, 0 + num);

if (surname != "") {

cout << surname << " ";

}

cout << name << " ";

if (patronymic != "") {

cout << patronymic << " ";

}

consoleCursorGoToXY(0, 1 + num);

}

int RecordNode::editRecord(LineNode\* line, int all, int now) {

int numFields = 1;

string arrFieldsStudent[1] = { "\_\_\_" };

checkLine checkFunks[1] = { checkLine\_Value };

string strLines[1] = { line->value };

Err arrErr[1] = { err\_Value };

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

int outFlag = fillFiedls(line, strLines[i], checkFunks[i], arrErr[i], arrFieldsStudent[i], numFields, 84, i + now - 1, 5, all - 1);

if (outFlag) {

consoleCursorGoToXY(0, 5 + all);

return 1;

}

}

consoleCursorGoToXY(0, 5 + all);

return 0;

}

void RecordNode::printRoom() {

cout << addZero(room, 10);

}

void RecordNode::printHeadRecordCard() {

system("CLS");

consoleCoutLetter("|", 0, 4);

consoleCoutLetter("|", 89, 4);

consoleCursorGoToXY(2, 0);

cout << "Ведомость№";

consoleCursorGoToXY(14, 0);

printRoom();

consoleCursorGoToXY(45, 0);

cout << "Дисциплина:";

cout << subject;

consoleCursorGoToXY(2, 1);

cout << "Дата проведения:";

cout << date;

consoleCursorGoToXY(45, 1);

cout << "Группа:";

cout << getGroup();

consoleCursorGoToXY(2, 2);

cout << "Преподователь:";

consoleCursorGoToXY(2, 3);

if (teacherSurname != "") {

cout << teacherSurname << " ";

}

cout << teacherName << " ";

if (teacherPatronymic != "") {

cout << teacherPatronymic << " ";

}

consoleCursorGoToXY(0, 4);

}

void RecordNode::printHeadRecordCardSNP() {

consoleCoutLetter("|", 0, 1, 4);

consoleCoutLetter("|", 89, 1, 4);

consoleCursorGoToXY(5, 4);

cout << "ФИО";

consoleCursorGoToXY(81, 4);

cout << "Оценка";

consoleCursorGoToXY(0, 5);

}

void RecordNode::printEditRecordOne(int num) {

consoleCoutLetter("|", 0, 1, 4 + num);

consoleCoutLetter("|", 89, 1, 4 + num);

consoleCursorGoToXY(2, 4 + num);

cout << num;

consoleCursorGoToXY(5, 4 + num);

if (surname != "") {

cout << surname << " ";

}

cout << name << " ";

if (patronymic != "") {

cout << patronymic << " ";

}

consoleCursorGoToXY(84, 4 + num);

cout << "\_\_\_";

consoleCursorGoToXY(84, 4 + num);

cout << value;

consoleCursorGoToXY(0, 5 + num);

}

void RecordNode::printStudentOne() {

cout << "| ";

cout << subject;

consoleSpace(27 - (2 + subject.length()));

cout << getValue(value);

consoleSpace(47 - 27 - getValue(value).length());

cout << "|";

cout << endl;

}

void RecordNode::printRecordOne(int num) {

consoleCoutLetter("|", 0, 1, 4 + num);

consoleCoutLetter("|", 89, 1, 4 + num);

consoleCursorGoToXY(2, 4 + num);

cout << num;

consoleCursorGoToXY(5, 4 + num);

if (surname != "") {

cout << surname << " ";

}

cout << name << " ";

if (patronymic != "") {

cout << patronymic << " ";

}

consoleCursorGoToXY(85, 4 + num);

cout << getValue(value);

consoleCursorGoToXY(0, 5 + num);

}

LineNode\* StudentNode::copyStudent() {

LineNode\* line = new LineNode;

line->type = 'S';

copyNumber(this->number, line->number);

line->checksumNumber = this->checksumNumber;

line->surname = this->surname;

line->name = this->name;

line->patronymic = this->patronymic;

line->institute = this->institute;

line->faculty = this->faculty;

line->training = this->training;

line->course = this->course;

line->group = this->group;

line->payment = this->payment;

line->year = this->year;

line->department = this->department;

line->date = this->date;

line->gender = this->gender;

line->address = this->address;

line->status = this->status;

line->checksumData = this->checksumData;

return line;

}

LineNode\* RecordNode::copyRecord() {

LineNode\* line = new LineNode;

line->type = 'R';

copyNumber(line->number, this->number);

line->checksumNumber = this->checksumNumber;

line->surname = this->surname;

line->name = this->name;

line->patronymic = this->patronymic;

line->institute = this->institute;

line->faculty = this->faculty;

line->training = this->training;

line->course = this->course;

line->group = this->group;

line->payment = this->payment;

line->year = this->year;

line->department = this->department;

line->date = this->date;

line->subject = this->subject;

line->teacherSurname = this->teacherSurname;

line->teacherName = this->teacherName;

line->teacherPatronymic = this->teacherPatronymic;

line->value = this->value;

line->room = this->room;

line->checksumData = this->checksumData;

return line;

}

StudentNode\* StudentNode::nextNode() {

return this->next;

}

RecordNode\* RecordNode::nextNode() {

return this->next;

}

void InstituteNode::resetSM() {

this->nodeSuccess = 0;

this->nodeMistake = 0;

this->operationSuccess = 0;

this->operationMistake = 0;

}

int InstituteNode::checkUniqueSNP(LineNode\* line) {

StudentNode\* node = headStudentsList;

while (node != nullptr) {

if (compareSNP(node, line) != 0) {

return compareSNP(node, line); //В данной группе уже есть студент с таким ФИО

}

node = node->next;

}

return 0;

}

int InstituteNode::compareSNP(StudentNode\* node, LineNode\* line) {

if ((node->institute == line->institute) && (node->faculty == line->faculty) && (node->course == line->course) && (node->group == line->group)) {

if ((node->surname != "") && (line->surname != "") && (node->patronymic != "") && (line->patronymic != "")) {

if (node->surname != line->surname) {

return 0;

}

if (node->name != line->name) {

return 0;

}

if (node->patronymic != line->patronymic) {

return 0;

}

return 1;

}

if (((node->surname == "") && (node->patronymic == "")) || ((line->surname == "") && (line->patronymic == ""))) {

if (node->name != line->name) {

return 0;

}

return 1;

}

if ((node->surname == "") || (line->surname == "")) {

if (node->name != line->name) {

return 0;

}

if (node->patronymic != line->patronymic) {

return 0;

}

return 1;

}

if ((node->patronymic == "") || (line->patronymic == "")) {

if (node->name != line->name) {

return 0;

}

if (node->surname != line->surname) {

return 0;

}

return 1;

}

return 1; //В данной группе уже есть студент с таким ФИО

}

else {

return 0;

}

}

int InstituteNode::checkUniqueGroup(LineNode\* line) {

StudentNode\* node = headStudentsList;

while (node != nullptr) {

if (compareGroup(node, line) != 0) {

return compareGroup(node, line); //Номера совпали

}

node = node->next;

}

return 0;

}

int InstituteNode::compareGroup(StudentNode\* node, LineNode\* line) {

if ((node->institute == line->institute) && (node->faculty == line->faculty) && (node->course == line->course) && (node->group == line->group)) {

if (node->training != line ->training) {

return 1; //Некоторые данные у одинаковых групп не совпали

}

if (node->payment != line->payment) {

return 2; //Некоторые данные у одинаковых групп не совпали

}

if (node->year != line->year) {

return 3; //Некоторые данные у одинаковых групп не совпали

}

if (node->department != line->department) {

return 4; //Некоторые данные у одинаковых групп не совпали

}

return 0;

}

return 0;

}

int InstituteNode::checkUniqueNumber(string\* str) {

StudentNode\* node = headStudentsList;

while (node != nullptr) {

if ((compareNumber(node->number, str)) == 1) {

return 1; //Номера совпали

}

node = node->next;

}

return 0;

}

int InstituteNode::compareNumber(string\* str1, string\* str2) {

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

if (str1[i][j] != str2[i][j]) {

return 0; //Не равны

}

}

}

return 1;

}

int InstituteNode::checkUniqueRoom(LineNode\* line) {

RecordNode\* node = headRecordsList;

while (node != nullptr) {

if ((compareRoom(node, line)) != 0) {

return 1; //Разные ведомости

}

node = node->next;

}

return 0;

}

int InstituteNode::compareRoom(RecordNode\* node, LineNode\* line) {

if (line->room == node->room) {

if (line->date != node->date) {

return 1;

}

if (line->subject != node->subject) {

return 2;

}

if (line->teacherSurname != node->teacherSurname) {

return 3;

}

if (line->teacherName != node->teacherName) {

return 4;

}

if (line->teacherPatronymic != node->teacherPatronymic) {

return 5;

}

}

return 0; //Ведомости равны

}

string InstituteNode::generateRecordRoom() {

RecordNode\* node = headRecordsList;

int num = 0;

while (node != nullptr) {

if (stoi(node->room) > num) {

num = stoi(node->room);

}

node = node->next;

}

return to\_string(num + 1);

}

void InstituteNode::generateNumber(LineNode\* line) {

string surname = toBigRus(line->surname);

for (int i = 0; i < 4; i++) {

if (surname[i] == '\0') {

break;

}

line->number[0][i] = surname[i];

}

string name = toBigRus(line->name);

for (int i = 0; i < 4; i++) {

if (name[i] == '\0') {

break;

}

line->number[1][i] = name[i];

}

string patronymic = toBigRus(line->patronymic);

for (int i = 0; i < 4; i++) {

if (patronymic[i] == '\0') {

break;

}

line->number[2][i] = patronymic[i];

}

tm ptm;

time\_t t = time(0);

gmtime\_s(&ptm, &t);

string date = "";

date = date + addZero(to\_string(ptm.tm\_mday), 2) + addZero(to\_string(ptm.tm\_mon + 1), 2) + addZero(to\_string(ptm.tm\_year + 1900), 4) + addZero(to\_string(ptm.tm\_min + 1), 2) + addZero(to\_string(ptm.tm\_sec + 1), 2);

for (int i = 0; i < 4; i++) {

line->number[3][i] = date[i];

}

for (int i = 0; i < 4; i++) {

line->number[4][i] = date[i + 4];

}

for (int i = 0; i < 4; i++) {

line->number[5][i] = date[i + 8];

}

string str = "";

do {

str = addZero(to\_string(rand() % 10000), 4);

for (int i = 0; i < 4; i++) {

line->number[6][i] = str[i];

}

str = addZero(to\_string(rand() % 10000), 4);

for (int i = 0; i < 4; i++) {

line->number[7][i] = str[i];

}

} while (checkUniqueNumber(line->number) == 1);

}

StudentNode\* InstituteNode::findStudent(LineNode\* line) {

StudentNode\* node = headStudentsList;

while (node != nullptr) {

if ((line->surname == node->surname) && (line->name == node->name) && (line->patronymic == node->patronymic) && (line->institute == node->institute) && (line->faculty == node->faculty) && (line->course == node->course) && (line->group == node->group)) {

return node;

}

node = node->next;

}

return nullptr;

}

RecordNode\* InstituteNode::findRecord(LineNode\* line) {

RecordNode\* node = headRecordsList;

while (node != nullptr) {

if (line->room == node->room) {

return node;

}

node = node->next;

}

return nullptr;

}

StudentNode\* InstituteNode::findGroup(LineNode\* line) {

StudentNode\* node = headStudentsList;

while (node != nullptr) {

if ((line->institute == node->institute) && (line->faculty == node->faculty) && (line->course == node->course) && (line->group == node->group)) {

return node;

}

node = node->next;

}

return nullptr;

}

void InstituteNode::add(LineNode\* line) {

if (line->getErr() == 0) {

line->errorUnique[0] = checkUniqueGroup(line);

if (line->errorUnique[0] == 0) {

if (line->type == 'S') {

addStudent(line);

return;

}

if (line->type == 'R') {

addRecord(line);

return;

}

}

else {

addFail(line);

return;

}

}

else {

addFail(line);

return;

}

return;

}

void InstituteNode::addStudent(LineNode\* line) {

line->errorUnique[1] = checkUniqueNumber(line->number);

if (line->errorUnique[1] == 0) {

line->errorUnique[2] = checkUniqueSNP(line);

if (line->errorUnique[2] == 0) {

nodeSuccess++;

StudentNode\* newStydent = new StudentNode(line);

//Добавление в алфавитном порядке снизу

if (headStudentsList == nullptr) {

headStudentsList = newStydent;

endStudentsList = newStydent;

}

else {

StudentNode\* node = endStudentsList;

string FIO1 = "";

string FIO2 = "";

FIO1 = newStydent->surname + newStydent->name + newStydent->patronymic;

FIO2 = node->surname + node->name + node->patronymic;

bool comp = 0;

while (true) {

if (compareString(toSmallRus(FIO1), toSmallRus(FIO2)) != 2) {

comp = 1;

break;

}

if (node->prev != nullptr) {

node = node->prev;

FIO2 = node->surname + node->name + node->patronymic;

}

else {

break;

}

}

if (!comp) { //Остановка из-за того, что nullptr

addStartStudent(newStydent);

}

else {

addAfterStudent(node, newStydent);

}

}

return;

}

}

addFail(line);

}

void InstituteNode::addRecord(LineNode\* line) {

line->errorUnique[3] = checkUniqueRoom(line);

if (line->errorUnique[3] == 0) {

nodeSuccess++;

RecordNode\* newRecord = new RecordNode(line);

//Добавление в алфавитном порядке снизу

if (headRecordsList == nullptr) {

headRecordsList = newRecord;

endRecordsList = newRecord;

}

else {

RecordNode\* node = endRecordsList;

string FIO1 = "";

string FIO2 = "";

FIO1 = newRecord->surname + newRecord->name + newRecord->patronymic;

FIO2 = node->surname + node->name + node->patronymic;

bool comp = 0;

while (true) {

if (compareString(toSmallRus(FIO1), toSmallRus(FIO2)) != 2) {

comp = 1;

break;

}

if (node->prev != nullptr) {

node = node->prev;

FIO2 = node->surname + node->name + node->patronymic;

}

else {

break;

}

}

if (!comp) { //Остановка из-за того, что nullptr

addStartRecord(newRecord);

}

else {

addAfterRecord(node, newRecord);

}

}

return;

}

addFail(line);

}

void InstituteNode::addFail(LineNode\* line) {

nodeMistake++;

if (failList == nullptr) {

failList = line;

lastFailList = line;

}

else {

lastFailList->next = line;

lastFailList = line;

}

return;

}

void InstituteNode::deleteAll() {

cout << endl;

cout << "Начало удачения данных" << endl;

StudentNode\* nodeS = headStudentsList;

while (nodeS != nullptr) {

nodeS = headStudentsList->next;

delete headStudentsList;

headStudentsList = nodeS;

nodeSuccess++;

}

RecordNode\* nodeR = headRecordsList;

while (nodeR != nullptr) {

nodeR = headRecordsList->next;

delete headRecordsList;

headRecordsList = nodeR;

nodeSuccess++;

}

LineNode\* nodeF = failList;

while (nodeF != nullptr) {

nodeF = failList->next;

delete failList;

failList = nodeF;

nodeSuccess++;

}

cout << "Удаление завершено, удалено " << nodeSuccess << " " << worlEnd\_запись[getWordEnd(nodeSuccess)] << " (в том числе некорректных)" << endl;

resetSM();

}

void InstituteNode::deleteOneStydent(string\* number) {

StudentNode\* node = headStudentsList;

while (node != nullptr) {

if (compareNumber(number, node->number) == 1) {

if (node != nullptr) {

if (node == headStudentsList) {

headStudentsList = node->next;

}

if (node == endStudentsList) {

endStudentsList = node->prev;

}

if (node->next != nullptr) {

node->next->prev = node->prev;

}

if (node->prev != nullptr) {

node->prev->next = node->next;

}

delete node;

}

return;

}

node = node->next;

}

}

void InstituteNode::deleteOneStydent(StudentNode\* node) {

if (node != nullptr) {

if (node == headStudentsList) {

headStudentsList = node->next;

}

if (node == endStudentsList) {

endStudentsList = node->prev;

}

if (node->next != nullptr) {

node->next->prev = node->prev;

}

if (node->prev != nullptr) {

node->prev->next = node->next;

}

delete node;

}

}

void InstituteNode::deleteOneRecord(RecordNode\* node) {

if (node != nullptr) {

if (node == headRecordsList) {

headRecordsList = node->next;

}

if (node == endRecordsList) {

endRecordsList = node->prev;

}

if (node->next != nullptr) {

node->next->prev = node->prev;

}

if (node->prev != nullptr) {

node->prev->next = node->next;

}

delete node;

}

}

void InstituteNode::deleteRecord(LineNode\* line) {

RecordNode\* node = headRecordsList;

while (node != nullptr) {

if (node->room == line->room) {

deleteOneRecord(node);

node = headRecordsList;

}

else {

node = node->next;

}

}

}

void InstituteNode::deleteGroup(LineNode\* line) {

StudentNode\* node = headStudentsList;

while (node != nullptr) {

if ((node->institute == line->institute) && (node->faculty == line->faculty) && (node->course == line->course) && (node->group == line->group)) {

deleteOneStydent(node);

node = headStudentsList;

}

else {

node = node->next;

}

}

}

void InstituteNode::addStartStudent(StudentNode\* newStydent) {

headStudentsList->prev = newStydent;

newStydent->next = headStudentsList;

headStudentsList = newStydent;

}

void InstituteNode::addAfterStudent(StudentNode\* node, StudentNode\* newStydent) {

newStydent->prev = node;

newStydent->next = node->next;

node->next = newStydent;

if (endStudentsList == node) {

endStudentsList = newStydent;

}

}

void InstituteNode::addStartRecord(RecordNode\* newRecord) {

headRecordsList->prev = newRecord;

newRecord->next = headRecordsList;

headRecordsList = newRecord;

}

void InstituteNode::addAfterRecord(RecordNode\* node, RecordNode\* newRecord) {

newRecord->prev = node;

newRecord->next = node->next;

node->next = newRecord;

if (endRecordsList == node) {

endRecordsList = newRecord;

}

}

void InstituteNode::printLineStudents() {

cout << endl;

cout << "Список студентов: " << endl;

StudentNode\* node = headStudentsList;

while (node != nullptr) {

node->printLineGeneral();

cout << " Рожд." << node->date << " Пол:" << node->gender << " Адрес:" << node->address;

cout << " ";

cout << "(" << getStatus(node->status) << ")";

cout << endl;

node = node->next;

}

if (headStudentsList == nullptr) {

cout << "Список студентов пуст" << endl;

}

}

void InstituteNode::printLineRecords() {

cout << endl;

cout << "Список ведомостей: " << endl;

RecordNode\* node = headRecordsList;

while (node != nullptr) {

node->printLineGeneral();

cout << " Провед." << node->date << " Предмет:" << node->subject << " Оценка:";

cout << getValue(node->value);

cout << " Преподователь:";

if (node->teacherSurname != "") {

cout << node->teacherSurname << " ";

}

cout << node->teacherName << " ";

if (node->teacherPatronymic != "") {

cout << node->teacherPatronymic << " ";

}

cout << " Номер ведомости №" << node->room << endl;

node = node->next;

}

if (headRecordsList == nullptr) {

cout << "Список ведомостей пуст" << endl;

}

}

void InstituteNode::printFailList(int num) {

LineNode\* node = failList;

while (node != nullptr) {

node->coutStr();

if (num) {

node->printErr(node);

}

node = node->next;

}

}

### Файл «Line.h»

#ifndef CW\_LINE\_H

#define CW\_LINE\_H

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

using namespace std;

struct Node; //Узел

struct LineNode; //Линия

string getStringBeforeSep(string& strLine, LineNode\* line);

int checkLine\_Lenght(string& strLine, LineNode\* line);

int checkLine\_Syntax(string& strLine, LineNode\* line);

int checkLine\_Type(string& strLine, LineNode\* line);

int checkLine\_AmountData(string& strLine, LineNode\* line);

int checkLine\_Number(string& strLine, LineNode\* line);

int checkLine\_ChecksumNumber(string& strLine, LineNode\* line);

int checkLine\_Surname(string& strLine, LineNode\* line);

int checkLine\_Name(string& strLine, LineNode\* line);

int checkLine\_Patronymic(string& strLine, LineNode\* line);

int checkLine\_Institute(string& strLine, LineNode\* line);

int checkLine\_Faculty(string& strLine, LineNode\* line);

int checkLine\_Training(string& strLine, LineNode\* line);

int checkLine\_Course(string& strLine, LineNode\* line);

int checkLine\_Group(string& strLine, LineNode\* line);

int checkLine\_Payment(string& strLine, LineNode\* line);

int checkLine\_Year(string& strLine, LineNode\* line);

int checkLine\_Department(string& strLine, LineNode\* line);

int checkLine\_Date(string& strLine, LineNode\* line);

int checkLine\_Gender(string& strLine, LineNode\* line);

int checkLine\_Address(string& strLine, LineNode\* line);

int checkLine\_Status(string& strLine, LineNode\* line);

int checkLine\_Subject(string& strLine, LineNode\* line);

int checkLine\_TeacherSurname(string& strLine, LineNode\* line);

int checkLine\_TeacherName(string& strLine, LineNode\* line);

int checkLine\_TeacherPatronymic(string& strLine, LineNode\* line);

int checkLine\_Value(string& strLine, LineNode\* line);

int checkLine\_Room(string& strLine, LineNode\* line);

int checkLine\_ChecksumData(string& strLine, LineNode\* line);

typedef int(\*checkLine)(string& strLine, LineNode\* line);

class Node {

public:

friend struct InstituteNode;

/\*Имя Фамилия Отчество Дата Год Мили Нано Ранд\*/

string\* number = new string[]{ " ", " ", " ", "2908", "1930", "0000", "0000", "0000" }; //Уникальный личный номер

string checksumNumber = ""; //Проверочный код личного номера

string surname = ""; //Фамилия

string name = ""; //Имя

string patronymic = ""; //Отчество

string institute = "М"; //Институт

string faculty = ""; //Факультет

string training = ""; //Форма обучения

string course = ""; //Курс

string group = ""; //Группа

string payment = ""; //Форма оплаты группы

string year = ""; //Год поступления

string department = ""; //Кафедра

string date = ""; //Дата

string checksumData = ""; //Контпольная сумма данных

virtual ~Node() {

if (number != nullptr) {

delete[] number;

}

};

string getGroup();

void printLineGeneral();

void writeGeneral(ostream& stream, Node\* node);

private:

};

class LineNode : public Node {

public:

friend struct InstituteNode;

int quantityErrorLine = 19;

int\* errorLine = new int[quantityErrorLine] { 0, //1 - недостаточная длинна

0, //2 - не соотвествует синтаксису

0, //3 - неизвестный тип записи

0, //4 - неверное количество данных

0, //5 - недопустимый номер

0, //6 - не сходится контрольна сумма числа

0, //7 - некорректная Фамилия

0, //8 - некорректное Имя

0, //9 - некорректное Отчество

0, //10 - некорректный институт

0, //11 - некорректный факультет

0, //12 - некорректная форма обучения

0, //13 - некорректный курс

0, //14 - некорректный номер группы

0, //15 - некорректный оплата группы

0, //16 - некорректный год поступления

0, //17 - некорректная кафедра

0, //18 - некорректная дата

0}; //19 - не сходится контрольна сумма данных

int quantityErrorStudent = 3;

int\* errorStudent = new int[quantityErrorStudent] { 0, //1 - некорректный пол

0, // 2 - некорректный адрес

0}; // 3 - некорректный статус обучения

int quantityErrorRecord = 6;

int\* errorRecord = new int[quantityErrorRecord] {0, //1 - некорректный предмет

0, //2 - некорректная Фамилия учителя

0, //3 - некорректное Имя учителя

0, //4 - некорректное Отчество учителя

0, //5 - некорректная оценка

0}; //6 - некорректный номер ведомости

int quantityErrorUnique = 4;

int\* errorUnique = new int[quantityErrorUnique] {0, //1 - у одинаковых групп разные данные

0, //2 - уникальный номер уже существует

0, //3 - в этой группе уже есть студент с таким ФИО

0}; //4 - у одинаковых ведомостей разные данные

string arrFields[7] = { "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "\_\_\_\_\_\_\_\_\_\_", "\_\_\_\_", "\_\_\_", "\_\_", "\_", ""};

char type = 'E'; //S - студент, R - ведомость

string gender = "С"; //Пол

string address = ""; //Адрес

string status = "1"; //Статус обучения

string subject = ""; //Предмет

string teacherSurname = ""; //Фамилия учителя

string teacherName = ""; //Имя учителя

string teacherPatronymic = ""; //Отчество учителя

string value = "-"; //Оценка

string room = ""; //Уникальный номер ведомости

char sep = ';';

int amountDataThis = 0;

int amountDataMinStudent = 19;

int amountDataMinRecord = 22;

int amountDataMin = min(this->amountDataMinStudent, this->amountDataMinRecord);

int lenStrMin = 4 + this->amountDataMin + 8 \* 5 - 1 + 6 \* 2;

LineNode() {

}

~LineNode() {

if (errorLine != nullptr) {

delete[] errorLine;

}

};

void coutError();

void setErr();

void resetErr();

bool getErr();

void coutErr();

void createStr(string str);

void coutStr();

string getStr();

void createLen(int len);

void addLen(int num);

void subLen(int num);

int getLen();

LineNode\* copyLineNode();

void printErr(LineNode\* node);

void coutCreateStudentCart(string arrFields[], int size, string strLines[]);

void coutFindStudentCart(string arrFields[], int size);

void coutCreateRecordCart(string arrFields[], int size1);

void coutFindGroupCart(string arrFields[], int size);

void coutFindRecordCart(string arrFields[], int size1);

int fillStydent(LineNode\* line, int operatinFlag = 1);

int findStydent(LineNode\* line);

int fillRecord(LineNode\* line);

int findRecord(LineNode\* line);

int findGroup(LineNode\* line);

LineNode\* nextNode(LineNode\* node);

private:

bool err = 0; //Была ли ошибка

string str = "";

int len = 0;

LineNode\* next = nullptr;

};

int fillFiedls(LineNode\* line, string strLine, checkLine checkFunk, Err checkErr, string field, int numFields, short w, short i, short hStart = 0, short down = 0);

#endif

### Файл «Line.cpp»

#include "Struct.h"

using namespace std;

string getStringBeforeSep(string& strLine, LineNode\* line) {

deleteSpace(strLine);

int i = 0;

while ((strLine[i] != '\0') && (strLine[i] != ';')) {

i++;

}

string str = strLine.substr(0, i);

line->subLen(i);

if (strLine[i] != '\0') {

strLine = strLine.substr(i + 1, strLine.length() - i - 1);

deleteSpace(strLine);

}

else {

strLine = "";

}

deleteSpace(str);

return str;

}

int checkLine\_Lenght(string& strLine, LineNode\* line) {

while (strLine[line->getLen()] != '\0') {

if (strLine[line->getLen()] == line->sep) {

line->amountDataThis++;

}

line->addLen(1);

}

if (line->getLen() < line->lenStrMin) {

return 1;

}

return 0;

}

int checkLine\_Syntax(string& strLine, LineNode\* line) {

if ((strLine[0] != '[') || (strLine[2] != ']') || (strLine[3] != ':') || (strLine[8] != '-') || (strLine[13] != '-') || (strLine[18] != '-') || (strLine[23] != '-') || (strLine[28] != '-') || (strLine[33] != '-') || (strLine[38] != '-') || (strLine[43] != ';') || (strLine[50] != ';')) {

return 1;

}

return 0;

}

int checkLine\_Type(string& strLine, LineNode\* line) {

if (strLine[1] == 'S') {

line->type = 'S';

return 0;

}

if (strLine[1] == 'R') {

line->type = 'R';

return 0;

}

return 1;

}

int checkLine\_AmountData(string& strLine, LineNode\* line) {

if (line->type == 'S') {

if (line->amountDataThis != line->amountDataMinStudent - 1) {

return 1;

}

return 0;

}

if (line->type == 'R') {

if (line->amountDataThis != line->amountDataMinRecord - 1) {

return 2;

}

return 0;

}

if (line->amountDataThis != line->amountDataMin - 1) {

return 3;

}

}

int checkLine\_Number(string& strLine, LineNode\* line) {

string number = getStringBeforeSep(strLine, line);

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

//cout << number[4 + i \* 5 + j] << " " << checkNumber(number[4 + i \* 5 + j]) << " ";

if ((checkNumber(number[4 + i \* 5 + j])) == 1) {

line->number[i][j] = number[4 + i \* 5 + j];

}

else {

return 1;

}

}

}

return 0;

}

int checkLine\_ChecksumNumber(string& strLine, LineNode\* line) {

string number;

string checksumNumber = getStringBeforeSep(strLine, line);

if (checksumNumber.length() != 6) {

return 1;

}

number = getNumber(line->number);

//cout << getChecksum(number) << endl;

if (getChecksum(number) != (checksumNumber)) {

return 1;

}

line->checksumNumber = checksumNumber;

return 0;

}

int checkLine\_Surname(string& strLine, LineNode\* line) {

string surname = getStringBeforeSep(strLine, line);

if (surname != "") {

if (isBigRus(surname[0]) == 0) {

return 1;

}

int i = 1;

while (surname[i] != '\0') {

if ((isSmallRus(surname[i]) != 1) && (isBigRus(surname[i]) != 1) && (surname[i] != '-') && (surname[i] != ' ')) {

return 2;

}

i++;

}

line->surname = surname;

return 0;

}

return 0;

}

int checkLine\_Name(string& strLine, LineNode\* line) {

string name = getStringBeforeSep(strLine, line);

if (name == "") {

return 1;

}

if (isBigRus(name[0]) == 0) {

return 2;

}

int i = 1;

while (name[i] != '\0') {

if ((isSmallRus(name[i]) != 1) && (isBigRus(name[i]) != 1) && (name[i] != '-') && (name[i] != ' ')) {

return 3;

}

i++;

}

line->name = name;

return 0;

}

int checkLine\_Patronymic(string& strLine, LineNode\* line) {

string patronymic = getStringBeforeSep(strLine, line);

if (patronymic != "") {

if (isBigRus(patronymic[0]) == 0) {

return 1;

}

int i = 1;

while (patronymic[i] != '\0') {

if ((isSmallRus(patronymic[i]) != 1) && (isBigRus(patronymic[i]) != 1) && (patronymic[i] != '-') && (patronymic[i] != ' ')) {

return 2;

}

i++;

}

line->patronymic = patronymic;

return 0;

}

return 0;

}

int checkLine\_Institute(string& strLine, LineNode\* line) {

string institute = getStringBeforeSep(strLine, line);

if (institute.length() != 1) {

return 1;

}

if ((institute[0] != 'М') && (institute[0] != 'M')) {

return 2;

}

line->institute = "М";

return 0;

}

int checkLine\_Faculty(string& strLine, LineNode\* line) {

string faculty = getStringBeforeSep(strLine, line);

if ((faculty.length() < 1) || (2 < faculty.length())) {

return 1;

}

int i = 0;

while (faculty[i] != '\0') {

if (isDigit(faculty[i]) == 0) {

return 2;

}

i++;

}

int facultyDigit = stoi(faculty);

if ((facultyDigit < 1) || (facultyDigit > 14)) {

return 3;

}

line->faculty = faculty;

return 0;

}

int checkLine\_Training(string& strLine, LineNode\* line) {

string training = getStringBeforeSep(strLine, line);

if (training.length() == 1) {

if ((training[0] == 'О') || (training[0] == 'O') || (training[0] == 'З')) {

if (training[0] == 'З') {

line->training = "З";

return 0;

}

else {

line->training = "О";

return 0;

}

}

return 1;

}

if ((training == "Оз") || (training == "Oз")) {

line->training = "Оз";

return 0;

}

return 2;

}

int checkLine\_Course(string& strLine, LineNode\* line) {

string course = getStringBeforeSep(strLine, line);

if (course.length() != 1) {

return 1;

}

if (isDigit(course[0]) == 0) {

return 2;

}

int courseDigit = stoi(course);

if ((courseDigit < 0) || (8 < courseDigit)) {

return 3;

}

line->course = course;

return 0;

}

int checkLine\_Group(string& strLine, LineNode\* line) {

string group = getStringBeforeSep(strLine, line);

if ((group.length() != 1) && (group.length() != 2)) {

return 1;

}

int i = 0;

while (group[i] != '\0') {

if (isDigit(group[i]) == 0) {

return 2;

}

i++;

}

int groupDigit = stoi(group);

if (groupDigit == 0) {

return 3;

}

if ((group.length() == 2) && (group[0] == '0')) {

group = group[1];

}

line->group = group;

return 0;

}

int checkLine\_Payment(string& strLine, LineNode\* line) {

string payment = getStringBeforeSep(strLine, line);

if (payment.length() == 1) {

if (payment[0] == 'А') {

line->payment = "А";

return 0;

}

if (payment[0] == 'М') {

line->payment = "М";

return 0;

}

if (payment[0] == 'Б') {

line->payment = "Б";

return 0;

}

if (payment[0] == 'С') {

line->payment = "С";

return 0;

}

return 1;

}

if (payment == "Бк") {

line->payment = "Бк";

return 0;

}

if (payment == "Ак") {

line->payment = "Ак";

return 0;

}

if (payment == "Мк") {

line->payment = "Мк";

return 0;

}

if (payment == "Ск") {

line->payment = "Ск";

return 0;

}

return 2;

}

int checkLine\_Year(string& strLine, LineNode\* line) {

string year = getStringBeforeSep(strLine, line);

if (year.length() != 4) {

return 1;

}

for (int i = 0; i < 4; i++) {

if (isDigit(year[i]) == 0) {

return 2;

}

}

int yearDigit = stoi(year);

if (yearDigit < 1930) {

return 3;

}

line->year = year;

return 0;

}

int checkLine\_Department(string& strLine, LineNode\* line) {

string department = getStringBeforeSep(strLine, line);

if (department.length() != 3) {

return 1;

}

for (int i = 0; i < 3; i++) {

if (isDigit(department[i]) == 0) {

return 2;

}

}

int departmentDigit = stoi(department);

if (departmentDigit == 0) {

return 3;

}

line->department = department;

return 0;

}

int checkLine\_Date(string& strLine, LineNode\* line) {

string date = getStringBeforeSep(strLine, line);

if (date.length() != 10) {

return 1;

}

if ((date[2] != '.') || (date[5] != '.')) {

return 2;

}

if ((isDigit(date[0]) == 0) || (isDigit(date[1]) == 0) || (isDigit(date[3]) == 0) || (isDigit(date[4]) == 0) || (isDigit(date[6]) == 0) || (isDigit(date[7]) == 0) || (isDigit(date[8]) == 0) || (isDigit(date[9]) == 0)) {

return 3;

}

string date1str = date.substr(6, 4);

int date1num = stoi(date1str);

if (line->type == 'R') {

if ((date1num < 1930) || (date1num > 2023)) {

return 4;

}

}

if (line->type == 'S') {

if (date1num > 2023) {

return 4;

}

}

string date2str = date.substr(3, 2);

int date2num = stoi(date2str);

if ((date2num < 1) || (date2num > 12)) {

return 5;

}

string date3str = date.substr(0, 2);

int date3num = stoi(date3str);

if ((date3num < 1) || (date3num > 31)) {

return 6;

}

line->date = date;

return 0;

}

int checkLine\_Gender(string& strLine, LineNode\* line) {

string gender = getStringBeforeSep(strLine, line);

if (gender.length() != 1) {

return 1;

}

if ((gender[0] != 'М') && (gender[0] != 'M') && (gender[0] != 'Ж') && (gender[0] != 'С') && (gender[0] != 'C')) {

return 2;

}

if ((gender[0] == 'М') || (gender[0] == 'M')) {

line->gender = "М";

return 0;

}

if ((gender[0] == 'С') || (gender[0] == 'C')) {

line->gender = "С";

return 0;

}

line->gender = "Ж";

return 0;

}

int checkLine\_Address(string& strLine, LineNode\* line) {

string address = getStringBeforeSep(strLine, line);

if (address == "") {

return 1;

}

int i = 0;

while (address[i] != '\0') {

if ((isDigit(address[i]) == 0) && (isSmallRus(address[i]) == 0) && (isBigRus(address[i]) == 0) && (address[i] != ' ') && (address[i] != '-') && (address[i] != '.') && (address[i] != ',')) {

cout << address << " " << i << endl;

return 2;

}

i++;

}

line->address = address;

return 0;

}

int checkLine\_Status(string& strLine, LineNode\* line) {

string status = getStringBeforeSep(strLine, line);

if (status.length() != 1) {

return 1;

}

if (isDigit(status[0]) == 0) {

return 2;

}

int statusDigit = stoi(status);

if ((statusDigit < 0) || (3 < statusDigit)) {

return 3;

}

line->status = status;

return 0;

}

int checkLine\_Subject(string& strLine, LineNode\* line) {

string subject = getStringBeforeSep(strLine, line);

if (subject == "") {

return 1;

}

int i = 0;

while (subject[i] != '\0') {

if ((isDigit(subject[i]) == 0) && (isSmallRus(subject[i]) == 0) && (isBigRus(subject[i]) == 0) && (subject[i] != ' ') && (subject[i] != '-') && (subject[i] != '.') && (subject[i] != ',')) {

return 2;

}

i++;

}

line->subject = subject;

return 0;

}

int checkLine\_TeacherSurname(string& strLine, LineNode\* line) {

string surname = getStringBeforeSep(strLine, line);

if (surname != "") {

if (isBigRus(surname[0]) == 0) {

return 1;

}

int i = 1;

while (surname[i] != '\0') {

if ((isSmallRus(surname[i]) != 1) && (isBigRus(surname[i]) != 1) && (surname[i] != '-') && (surname[i] != ' ')) {

return 2;

}

i++;

}

line->teacherSurname = surname;

return 0;

}

return 0;

}

int checkLine\_TeacherName(string& strLine, LineNode\* line) {

string name = getStringBeforeSep(strLine, line);

if (name == "") {

return 1;

}

if (isBigRus(name[0]) == 0) {

return 2;

}

int i = 1;

while (name[i] != '\0') {

if ((isSmallRus(name[i]) != 1) && (isBigRus(name[i]) != 1) && (name[i] != '-') && (name[i] != ' ')) {

return 3;

}

i++;

}

line->teacherName = name;

return 0;

}

int checkLine\_TeacherPatronymic(string& strLine, LineNode\* line) {

string patronymic = getStringBeforeSep(strLine, line);

if (patronymic != "") {

if (isBigRus(patronymic[0]) == 0) {

return 1;

}

int i = 1;

while (patronymic[i] != '\0') {

if ((isSmallRus(patronymic[i]) != 1) && (isBigRus(patronymic[i]) != 1) && (patronymic[i] != '-') && (patronymic[i] != ' ')) {

return 2;

}

i++;

}

line->teacherPatronymic = patronymic;

return 0;

}

return 0;

}

int checkLine\_Value(string& strLine, LineNode\* line) {

string value = getStringBeforeSep(strLine, line);

if (value == "") {

line->value = "-";

return 0;

}

if (value.length() == 1) {

if (value[0] == '-') {

line->value = "-";

return 0;

}

if (isDigit(value[0]) == 0) {

return 1;

}

int valueDigit = stoi(value);

if ((valueDigit < 2) || (5 < valueDigit)) {

return 2;

}

line->value = value;

return 0;

}

if (value.length() == 3) {

if (value == "н/я") {

line->value = "н/я";

return 0;

}

if (value == "н/д") {

line->value = "н/д";

return 0;

}

if (value == "н/к") {

line->value = "н/к";

return 0;

}

}

return 3;

}

int checkLine\_Room(string& strLine, LineNode\* line) {

string room = getStringBeforeSep(strLine, line);

if (room == "") {

return 1;

}

int i = 0;

while (room[i] != '\0') {

if (isDigit(room[i]) == 0) {

return 2;

}

i++;

}

line->room = room;

return 0;

}

int checkLine\_ChecksumData(string& strLine, LineNode\* line) {

string data = "";

string checksumData = getStringBeforeSep(strLine, line);

if (checksumData.length() != 6) {

return 1;

}

data = data + line->type + getNumber(line->number) + line->checksumNumber + line->surname + line->name + line->patronymic + line->institute + line->faculty + line->training + line->course + line->group + line->payment + line->year + line->department + line->date;

if (line->type == 'S') {

data = data + line->gender + line->address + line->status;

}

if (line->type == 'R') {

data = data + line->subject + line->teacherSurname + line->teacherName + line->teacherPatronymic + line->value + line->room;

}

//cout << getChecksum(data) << endl;

if (getChecksum(data) != (checksumData)) {

return 1;

}

line->checksumData = checksumData;

return 0;

}

string Node::getGroup() {

string str = "";

str = str + institute + faculty + training + "-" + course + addZero(group, 2) + payment + "-" + year + " (каф." + department + ")";

return str;

}

void Node::printLineGeneral() {

cout << getGroup();

consoleSpace(29 - getGroup().length());

if (surname != "") {

cout << surname << " ";

}

cout << name << " ";

if (patronymic != "") {

cout << patronymic << " ";

}

}

void Node::writeGeneral(ostream& stream, Node\* node) {

stream << ";" << node->checksumNumber << ";" << node->surname << ";" << node->name << ";" << node->patronymic << ";" << node->institute << ";" << node->faculty << ";" << node->training << ";" << node->course << ";" << node->group << ";" << node->payment << ";" << node->year << ";" << node->department << ";" << node->date;

stream << ";";

}

void LineNode::coutError() {

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

cout << errorLine[i];

}

cout << " ";

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

cout << errorStudent[i];

}

cout << " ";

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

cout << errorRecord[i];

}

cout << " " << err << endl;

}

void LineNode::setErr() {

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

if (this->errorLine[i] != 0) { this->err = 1; }

}

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

if (this->errorStudent[i] != 0) { this->err = 1; }

}

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

if (this->errorRecord[i] != 0) { this->err = 1; }

}

}

void LineNode::resetErr() {

this->err = 0;

}

bool LineNode::getErr() {

return this->err;

}

void LineNode::coutErr() {

cout << this->err << endl;

}

void LineNode::createStr(string str) {

this->str = str;

}

void LineNode::coutStr() {

cout << this->str << endl;

}

string LineNode::getStr() {

return this->str;

}

void LineNode::createLen(int len) {

this->len = len;

}

void LineNode::addLen(int num) {

this->len = this->len + num;

}

void LineNode::subLen(int num) {

this->len = this->len - num;

}

int LineNode::getLen() {

return this->len;

}

LineNode\* LineNode::copyLineNode() {

LineNode\* line = new LineNode;

line->type = this->type;

copyNumber(line->number, this->number);

line->checksumNumber = this->checksumData;

line->surname = this->surname;

line->name = this->name;

line->patronymic = this->patronymic;

line->institute = this->institute;

line->faculty = this->faculty;

line->training = this->training;

line->course = this->course;

line->group = this->group;

line->payment = this->payment;

line->year = this->year;

line->department = this->department;

line->date = this->date;

line->gender = this->gender;

line->address = this->address;

line->status = this->status;

line->subject = this->subject;

line->teacherSurname = this->teacherSurname;

line->teacherName = this->teacherName;

line->teacherPatronymic = this->teacherPatronymic;

line->value = this->value;

line->room = this->room;

line->checksumData = this->checksumData;

return line;

}

void LineNode::printErr(LineNode\* node) {

Err err\_Total[] = { err\_Lenght, err\_Syntax, err\_Type, err\_AmountData, err\_Number, err\_ChecksumNumber, err\_Surname, err\_Name, err\_Patronymic, err\_Institute, err\_Faculty, err\_Training, err\_Course, err\_Group, err\_Payment, err\_Year, err\_Department, err\_Date, err\_ChecksumData };

Err err\_Student[] = { err\_Gender, err\_Address, err\_Status };

Err err\_Record[] = { err\_Subject, err\_TeacherSurname, err\_TeacherName, err\_TeacherPatronymic, err\_Value, err\_Room };

Err err\_Unique[] = { err\_UniqueGroup, err\_UniqueNumber, err\_UniqueSNA, err\_UniqueRoom };

string sep = " ";

for (int i = 0; i < node->quantityErrorLine; i++) {

if (node->errorLine[i] != 0) {

cout << sep;

err\_Total[i](node->errorLine[i]);

cout << endl;

}

}

for (int i = 0; i < node->quantityErrorStudent; i++) {

if (node->errorStudent[i] != 0) {

cout << sep;

err\_Student[i](node->errorStudent[i]);

cout << endl;

}

}

for (int i = 0; i < node->quantityErrorRecord; i++) {

if (node->errorRecord[i] != 0) {

cout << sep;

err\_Record[i](node->errorRecord[i]);

cout << endl;

}

}

for (int i = 0; i < node->quantityErrorUnique; i++) {

if (node->errorUnique[i] != 0) {

cout << sep;

err\_Unique[i](node->errorUnique[i]);

cout << endl;

}

}

}

/\*Функция считывания, обработки и записи полученного значения

Куда записывать, что было в начале, функция проверки корретности данных, функция печати ошибки о некорректности, заполняемое поле, кол-во заполняемых полей(для печати ошибки внизу), отспуп по ширине, какой по счёту(для понимая сколько от всего введённого отспуать вниз), сколько ещё взять вниз для заполнения, сколько ещё взять вниз для вывода ошибки\*/

int fillFiedls(LineNode\* line, string strLine, checkLine checkFunk, Err checkErr, string field, int numFields, short w, short i, short hStart, short down) {

string copeLine = "";

int run = 1;

while (run) { //Вводи данное поле, пока оно не станет корректным

consoleCursorGoToXY(w, i + hStart); //Печатаем редактируемое значение

cout << field;

consoleCursorGoToXY(w, i + hStart);

cout << strLine;

consoleCursorGoToXY(w, i + hStart);

int outFlag = getConsole(strLine, w, i + hStart, field.length()); //Считываем данные

copeLine = strLine;

consoleCursorGoToXY(0, numFields + hStart + down);

if (outFlag) {

return 1;

}

run = checkFunk(strLine, line); //Проверяем данные на корректномть

consoleClearLine();

consoleCursorGoToXY(0, numFields + hStart + down);

if (run != 0) { //Печатаем если ошибка

checkErr(run);

cout << endl;

}

}

consoleCursorGoToXY(w, i + hStart); //Вывод полученного

cout << field;

consoleCursorGoToXY(w, i + hStart);

cout << copeLine;

consoleCursorGoToXY(0, numFields + hStart + down); //Очистка после возможного вывода ошибок

consoleClearLine();

consoleCursorGoToXY(0, numFields + hStart + down);

return 0;

}

void LineNode::coutCreateStudentCart(string arrFields[], int size1, string strLines[]) {

//cout << "| Отчество 12345678901234567890 |" << endl;

string arrFillIn[] = { "Студент", "Фамилия", "Имя", "Отчество", "Институт", "Факультет", "Форма обучения", "Курс", "Группа", "Коммерц. группы", "Год поступления", "Кафедра", "Дата рождения", "Пол", "Адрес", "Статус обучения" };

consoleCoutLetter("|", 0, size1 + 1);

consoleCoutLetter("|", 47, size1 + 1);

consoleCoutMesage(arrFillIn, 2, size1 + 1);

consoleCoutMesage(arrFields, 19, size1, 1);

consoleCoutMesage(strLines, 19, size1, 1);

}

void LineNode::coutFindStudentCart(string arrFields[], int size1) {

string arrFillIn[] = { "Студент", "Фамилия", "Имя", "Отчество", "Институт", "Факультет", "Курс", "Группа" };

consoleCoutLetter("|", 0, size1 + 1);

consoleCoutLetter("|", 47, size1 + 1);

consoleCoutMesage(arrFillIn, 2, size1 + 1);

consoleCoutMesage(arrFields, 19, size1, 1);

}

void LineNode::coutCreateRecordCart(string arrFields[], int size1) {

string arrFillIn[] = { "Ведомость", "Предмет", "Дата проведения", "Фамилия пр.", "Имя пр.", "Отчество пр." };

consoleCoutLetter("|", 0, size1 + 1);

consoleCoutLetter("|", 47, size1 + 1);

consoleCoutMesage(arrFillIn, 2, size1 + 1);

consoleCoutMesage(arrFields, 19, size1, 1);

}

void LineNode::coutFindGroupCart(string arrFields[], int size1) {

string arrFillIn[] = { "Группа", "Институт", "Факультет", "Курс", "Группа" };

consoleCoutLetter("|", 0, size1 + 1);

consoleCoutLetter("|", 47, size1 + 1);

consoleCoutMesage(arrFillIn, 2, size1 + 1);

consoleCoutMesage(arrFields, 19, size1, 1);

}

void LineNode::coutFindRecordCart(string arrFields[], int size1) {

string arrFillIn[] = { "Ведомость" };

consoleCoutLetter("|", 0, size1);

consoleCoutLetter("|", 47, size1);

consoleCoutMesage(arrFillIn, 2, size1);

consoleCoutMesage(arrFields, 19, size1);

}

int LineNode::fillStydent(LineNode\* line, int operatinFlag) { //Функция заполнения полей для создания/редактирования студента

this->type = 'S';

int numFields = 15;

string arrFieldsStudent[15] = { arrFields[0], arrFields[0], arrFields[0], arrFields[5], arrFields[4], arrFields[4], arrFields[5], arrFields[4], arrFields[4], arrFields[2], arrFields[3], arrFields[1], arrFields[5], arrFields[0], arrFields[5] };

checkLine checkFunks[15] = { checkLine\_Surname, checkLine\_Name, checkLine\_Patronymic, checkLine\_Institute, checkLine\_Faculty, checkLine\_Training , checkLine\_Course, checkLine\_Group, checkLine\_Payment, checkLine\_Year, checkLine\_Department, checkLine\_Date, checkLine\_Gender, checkLine\_Address, checkLine\_Status };

string strLines[15] = { line->surname, line->name, line->patronymic, line->institute, line->faculty, line->training, line->course,line->group, line->payment, line->year, line->department, line->date, line->gender, line->address, line->status };

Err arrErr[15] = { err\_Surname, err\_Name, err\_Patronymic, err\_Institute, err\_Faculty, err\_Training, err\_Course, err\_Group, err\_Payment, err\_Year, err\_Department, err\_Date, err\_Gender, err\_Address, err\_Status };

coutCreateStudentCart(arrFieldsStudent, numFields, strLines);

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

int outFlag = fillFiedls(line, strLines[i], checkFunks[i], arrErr[i], arrFieldsStudent[i], numFields, 19, i, 1);

if (outFlag) {

return 1;

}

}

consoleCursorGoToXY(0, numFields + 1);

return 0;

}

int LineNode::findStydent(LineNode\* line) { //Функция заполнения полей для поиска студента

int numFields = 7;

string arrFieldsStudent[7] = { arrFields[0], arrFields[0], arrFields[0], arrFields[5], arrFields[4], arrFields[5], arrFields[4] };

checkLine checkFunks[7] = { checkLine\_Surname, checkLine\_Name, checkLine\_Patronymic, checkLine\_Institute, checkLine\_Faculty , checkLine\_Course, checkLine\_Group };

string strLines[7] = { arrFields[6], arrFields[6], arrFields[6], arrFields[6], arrFields[6], arrFields[6], arrFields[6] };

Err arrErr[7] = { err\_Surname, err\_Name, err\_Patronymic, err\_Institute, err\_Faculty, err\_Course, err\_Group };

coutFindStudentCart(arrFieldsStudent, 7);

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

int outFlag = fillFiedls(line, strLines[i], checkFunks[i], arrErr[i], arrFieldsStudent[i], numFields, 19, i, 1);

if (outFlag) {

return 1;

}

}

consoleCursorGoToXY(0, numFields + 1);

return 0;

}

int LineNode::fillRecord(LineNode\* line) {//Функция заполнения полей для создания ведомости

this->type = 'R';

int numFields = 5;

string arrFieldsStudent[5] = { arrFields[0], arrFields[1], arrFields[0], arrFields[0], arrFields[0] };

checkLine checkFunks[5] = { checkLine\_Subject, checkLine\_Date, checkLine\_TeacherSurname, checkLine\_TeacherName, checkLine\_TeacherPatronymic };

string strLines[5] = { arrFields[6], arrFields[6], arrFields[6], arrFields[6], arrFields[6] };

Err arrErr[5] = { err\_Subject, err\_Date, err\_TeacherSurname, err\_TeacherName, err\_TeacherPatronymic };

coutCreateRecordCart(arrFieldsStudent, 5);

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

int outFlag = fillFiedls(line, strLines[i], checkFunks[i], arrErr[i], arrFieldsStudent[i], numFields, 19, i, 1);

if (outFlag) {

return 1;

}

}

consoleCursorGoToXY(0, numFields + 1);

return 0;

}

int LineNode::findRecord(LineNode\* line) { //Функция заполнения полей для создания нахождения

int numFields = 1;

string arrFieldsStudent[1] = { arrFields[1] };

checkLine checkFunks[1] = { checkLine\_Room };

string strLines[1] = { arrFields[6] };

Err arrErr[1] = { err\_Room };

coutFindRecordCart(arrFieldsStudent, 1);

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

int outFlag = fillFiedls(line, strLines[i], checkFunks[i], arrErr[i], arrFieldsStudent[i], numFields, 19, i);

if (outFlag) {

return 1;

}

}

consoleCursorGoToXY(0, numFields);

return 0;

}

int LineNode::findGroup(LineNode\* line) { //Функция заполнения полей для нахождения группы

int numFields = 4;

string arrFieldsStudent[4] = { arrFields[5], arrFields[4], arrFields[5], arrFields[4] };

checkLine checkFunks[4] = { checkLine\_Institute, checkLine\_Faculty , checkLine\_Course, checkLine\_Group };

string strLines[4] = { arrFields[6], arrFields[6], arrFields[6], arrFields[6] };

Err arrErr[4] = { err\_Institute, err\_Faculty, err\_Course, err\_Group };

coutFindGroupCart(arrFieldsStudent, 4);

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

int outFlag = fillFiedls(line, strLines[i], checkFunks[i], arrErr[i], arrFieldsStudent[i], numFields, 19, i, 1);

if (outFlag) {

return 1;

}

}

consoleCursorGoToXY(0, numFields + 1);

return 0;

}

LineNode\* LineNode::nextNode(LineNode\* node) {

return node->next;

}

### Файл «Console.h»

#ifndef CW\_CONSOLE\_H

#define CW\_CONSOLE\_H

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

#include "Char.h"

using namespace std;

#define kESC 27

#define kSPACE 32

#define kENTER 13

#define kBACKSPACE 8

#define k1 49

#define k2 50

#define k3 51

#define k4 52

#define k5 53

#define k6 54

#define k7 55

#define k8 56

#define k9 57

#define k0 58

string getLetter(char key);

void consoleCursorGoToXY(short x, short y);

void consoleCursorVisible(bool show);

void consoleCoutLetter(string let, short w, short h, short hStart = 0);

void consoleCoutMesage(string arr[], short w, short h, short hStart = 0);

void consoleClearLine();

void consoleSpace(int num);

int getConsole(string& strLine, short w, short h, int len, short hStart = 0);

#endif

### Файл «Console.cpp»

#include "Console.h"

using namespace std;

string getLetter(char key) {

switch (key) {

case -48:

key = \_getch();

switch (key) {

case -112:

return "А";

case -111:

return "Б";

case -110:

return "В";

case -109:

return "Г";

case -108:

return "Д";

case -107:

return "Е";

case -127:

return "Ё";

case -106:

return "Ж";

case -105:

return "З";

case -104:

return "И";

case -103:

return "Й";

case -102:

return "К";

case -101:

return "Л";

case -100:

return "М";

case -99:

return "Н";

case -98:

return "О";

case -97:

return "П";

case -96:

return "Р";

case -95:

return "С";

case -94:

return "Т";

case -93:

return "У";

case -92:

return "Ф";

case -91:

return "Х";

case -90:

return "Ц";

case -89:

return "Ч";

case -88:

return "Ш";

case -87:

return "Щ";

case -86:

return "Ъ";

case -85:

return "Ы";

case -84:

return "Ь";

case -83:

return "Э";

case -82:

return "Ю";

case -81:

return "Я";

case -80:

return "а";

case -79:

return "б";

case -78:

return "в";

case -77:

return "г";

case -76:

return "д";

case -75:

return "е";

case -74:

return "ж";

case -73:

return "з";

case -72:

return "и";

case -71:

return "й";

case -70:

return "к";

case -69:

return "л";

case -68:

return "м";

case -67:

return "н";

case -66:

return "о";

case -65:

return "п";

default:

return "";

}

case -47:

key = \_getch();

switch (key) {

case -111:

return "ё";

case -128:

return "р";

case -127:

return "с";

case -126:

return "т";

case -125:

return "у";

case -124:

return "ф";

case -123:

return "х";

case -122:

return "ц";

case -121:

return "ч";

case -120:

return "ш";

case -119:

return "щ";

case -118:

return "ъ";

case -117:

return "ы";

case -116:

return "ь";

case -115:

return "э";

case -114:

return "ю";

case -113:

return "я";

default:

return "";

}

case 32:

return " ";

case 44:

return ",";

case 45:

return "-";

case 46:

return ".";

case 47:

return "/";

case 48:

return "0";

case 49:

return "1";

case 50:

return "2";

case 51:

return "3";

case 52:

return "4";

case 53:

return "5";

case 54:

return "6";

case 55:

return "7";

case 56:

return "8";

case 57:

return "9";

default:

return "";

}

return "";

}

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

void consoleCursorGoToXY(short x, short y) {

SetConsoleCursorPosition(hStdOut, { x, y });

}

void consoleCursorVisible(bool show) {

CONSOLE\_CURSOR\_INFO structCursorInfo;

GetConsoleCursorInfo(hStdOut, &structCursorInfo);

structCursorInfo.bVisible = show;

SetConsoleCursorInfo(hStdOut, &structCursorInfo);

}

void consoleCoutLetter(string let, short w, short h, short hStart) { //Что выводить, сколько отступать, сколько выводить, сколько ещё вниз отступить

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

consoleCursorGoToXY(w, i + hStart);

cout << let;

}

}

void consoleCoutMesage(string arr[], short w, short h, short hStart) { //Что выводить, сколько отступать, сколько выводить, сколько ещё вниз отступить

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

consoleCursorGoToXY(w, i + hStart);

cout << arr[i];

}

}

void consoleClearLine() {

cout << " " << endl;

}

void consoleSpace(int num) {

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

cout << " ";

}

}

int getConsole(string& strLine, short w, short h, int maxLen, short hStart) { //Во что считывать, сколько отступать бок, сколько отступать низ, максимальная длинна считывания, сколько ещё вниз отступить

int len = strLine.length();

consoleCursorGoToXY(w + len, h + hStart); //Ставим курсор в зависимости от уже напечатанного

consoleCursorVisible(true);

char key;

bool run = 1;

while (run){

key = \_getch();

switch (key) {

case kESC: //Выход

consoleCursorVisible(false);

return 1;

case kENTER: //Ввод

deleteSpace(strLine);

consoleCursorVisible(false);

return 0;

case kBACKSPACE: { //Стирание

if (len > 0) {

len--;

strLine = strLine.substr(0, len);

consoleCursorGoToXY(w + len, h + hStart);

if (len >= maxLen) {

cout << " ";

}

else {

cout << "\_";

}

consoleCursorGoToXY(w + len, h + hStart);

}

}

break;

default: { //Считывание клавиши

if (len < maxLen) {

string sym = getLetter(key);

if (sym != "") {

strLine = strLine + sym;

if (sym == " ") {

cout << "\_";

}

else {

cout << sym;

}

len++;

}

}

}

}

}

### }Файл «Char.h»

#ifndef CW\_CHAR\_H

#define CW\_CHAR\_H

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

using namespace std;

int getWordEnd(int num);

void deleteSpace(string& str);

string addZero(string str, int num);

string getStatus(string val);

string getValue(string val);

string getChecksum(string str);

string getNumber(string\* str);

int compareString(string str1, string str2);

void copyNumber(string\* from, string\* to);

bool isDigit(char ch);

bool isBigRus(char ch);

bool isSmallRus(char ch);

bool checkNumber(char ch);

string toSmallRus(string str);

string toBigRus(string str);

#endif

### Файл «Char.cpp»

#include "Char.h"

using namespace std;

int getWordEnd(int num) {

if ((num % 10 <= 0) || (num % 10 >= 5)) {

return 0;

}

if ((num % 10 == 1)) {

return 1;

}

return 2;

//0 записей

//1 запись

//2 записи

//3 записи

//4 записи

//5 записей

//6 записей

//7 записей

//8 записей

//9 записей

}

void deleteSpace(string& str) {

if (str[0] == ' ') {

str.erase(str.find(' '), 1);

deleteSpace(str);

}

if (str.length() != 0) {

if (str[str.length() - 1] == ' ') {

str = str.substr(0, str.length() - 1);

deleteSpace(str);

}

}

return;

}

string addZero(string str, int num) {

if (str.length() < num) {

str = '0' + str;

str = addZero(str, num);

}

return str;

}

string getStatus(string val) {

if (val == "1") {

return "Продолжает обучение";

}

if (val == "2") {

return "В академическом отпуске";

}

if (val == "3") {

return "Закончил обучение";

}

if (val == "0") {

return "Отчислен";

}

}

string getValue(string val) {

if (val == "") {

return "-";

}

if (val == "-") {

return "-";

}

if (val == "0") {

return "";

}

if (val == "1") {

return "н/я";

}

if (val == "6") {

return "н/д";

}

if (val == "7") {

return "н/к";

}

else {

return val;

}

}

string getChecksum(string str) { //Функция получения контрольной суммы

if (str.size() % 2 == 1) {

str = str + 'm';

}

unsigned long long int sum = 0;

int mul;

int div;

for (int i = 0; i < str.size(); i = i + 2) { //Прибавляем к контрольной сумме результат попарного деления и умножения их значений

mul = str[i] \* str[i + 1];

div = str[i] / str[i + 1];

sum = sum + mul + div;

sum = sum % 1000000000000000000; //Предотвращаем переполнение

}

while (sum % 10 == 0) { //Убираем нули в конце

sum = sum / 10;

}

sum = sum % 1000000; //Оставляем 6 последних цифр

str = to\_string(sum);

if (str.length() < 6) {

str = addZero(str, 6); //При длине меньше 6 - добавляем значимые нули

}

return str;

}

string getNumber(string\* str) {

string number = "";

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

number = number + str[i][j];

}

}

return number;

}

int compareString(string str1, string str2) { //Сравнение строк в алфавитном порядке

int i = 0;

while ((str1[i] != '\0') && (str2[i] != '\0')) {

if (str1[i] > str2[i]) {

return 1; //Первая больше

}

if (str1[i] < str2[i]) {

return 2; //Вторая больше

}

i++;

}

if (str1[i] == '\0') {

return 2;

}

return 0; //Одинаковые

}

void copyNumber(string\* from, string\* to) {

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

to[i][j] = from[i][j];

}

}

}

bool isDigit(char ch) {

if (('0' <= ch) && (ch <= '9')) {

return 1;

}

return 0;

}

bool isBigRus(char ch) {

if ((('А' <= ch) && (ch <= 'Я')) || (ch == 'Ё')) {

return 1;

}

return 0;

}

bool isSmallRus(char ch) {

if ((('а' <= ch) && (ch <= 'я')) || (ch == 'ё')) {

return 1;

}

return 0;

}

bool checkNumber(char ch) {

if (isDigit(ch) == 1) {

return 1;

}

if (isBigRus(ch) == 0) {

return 1;

}

if (ch == ' ') {

return 1;

}

if (ch != '-') {

return 1;

}

return 0; //Номер не допустим

}

string toSmallRus(string str) {

int i = 0;

while (str[i] != '\0') {

if (str[i] == 'Ё') {

str[i] = 'ё';

}

else {

if (isBigRus(str[i]) == 1) {

str[i] = str[i] + 32;

}

}

i++;

}

return str;

}

string toBigRus(string str) {

int i = 0;

while (str[i] != '\0') {

if (str[i] == 'ё') {

str[i] = 'Ё';

}

else {

if (isSmallRus(str[i]) == 1) {

str[i] = str[i] - 32;

}

}

i++;

}

return str;

}

### Файл «Error.h»

#ifndef CW\_ERROR\_H

#define CW\_ERROR\_H

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

void err\_Lenght(int err);

void err\_Syntax(int err);

void err\_Type(int err);

void err\_AmountData(int err);

void err\_Number(int err);

void err\_ChecksumNumber(int err);

void err\_Surname(int err);

void err\_Name(int err);

void err\_Patronymic(int err);

void err\_Institute(int err);

void err\_Faculty(int err);

void err\_Training(int err);

void err\_Course(int err);

void err\_Group(int err);

void err\_Payment(int err);

void err\_Year(int err);

void err\_Department(int err);

void err\_Date(int err);

void err\_Gender(int err);

void err\_Address(int err);

void err\_Status(int err);

void err\_Subject(int err);

void err\_TeacherSurname(int err);

void err\_TeacherName(int err);

void err\_TeacherPatronymic(int err);

void err\_Value(int err);

void err\_Room(int err);

void err\_ChecksumData(int err);

void err\_UniqueGroup(int err);

void err\_UniqueNumber(int err);

void err\_UniqueSNA(int err);

void err\_UniqueRoom(int err);

typedef void (\*Err)(int err);

#endif

### Файл «Error.cpp»

#include "Error.h"

using namespace std;

string fileDam = ", файл повреждён";

void err\_Lenght(int err) {

if (err == 1) { cout << "Недостаточная длинна строки" << fileDam; return; }

cout << "Необработанная ошибка 1:" << err;

}

void err\_Syntax(int err) {

if (err == 1) { cout << "Несоотвествие синтаксису файла" << fileDam; return; }

cout << "Необработанная ошибка 2:" << err;

}

void err\_Type(int err) {

if (err == 1) { cout << "Некорректый тип записи" << fileDam; return; }

cout << "Необработанная ошибка 3:" << err;

}

void err\_AmountData(int err) {

if (err == 1) { cout << "Недостаточное количество данных" << fileDam; return; }

if (err == 2) { cout << "Недостаточное количество данных" << fileDam; return; }

if (err == 3) { cout << "Недостаточное количество данных" << fileDam; return; }

cout << "Необработанная ошибка 4:" << err;

}

void err\_Number(int err) {

if (err == 1) { cout << "Некорректный личный номер" << fileDam; return; }

cout << "Необработанная ошибка 5:" << err;

}

void err\_ChecksumNumber(int err) {

if (err == 1) { cout << "Не совпадает контрольная сумма личного номера" << fileDam; return; }

cout << "Необработанная ошибка 6:" << err;

}

void err\_Surname(int err) {

if (err == 1) { cout << "Фамилия начинается не с большой буквы"; return; }

if (err == 2) { cout << "В фамилии присутсвуют недопустимые символы"; return; }

cout << "Необработанная ошибка 7:" << err;

}

void err\_Name(int err) {

if (err == 1) { cout << "Отсутсвует имя"; return; }

if (err == 2) { cout << "Имя начинается не с большой буквы"; return; }

if (err == 3) { cout << "В имени присутствуют недопустимые символы"; return; }

cout << "Необработанная ошибка 8:" << err;

}

void err\_Patronymic(int err) {

if (err == 1) { cout << "Отчество начинается не с большой буквы"; return; }

if (err == 2) { cout << "В отчестве присутсвуют недопустимые символы"; return; }

cout << "Необработанная ошибка 9:" << err;

}

void err\_Institute(int err) {

if (err == 1) { cout << "Неправильная длинна значения института"; return; }

if (err == 2) { cout << "Неправильный институт"; return; }

cout << "Необработанная ошибка 10:" << err;

}

void err\_Faculty(int err) {

if (err == 1) { cout << "Недопустимая длинна значения факультета"; return; }

if (err == 2) { cout << "В номере факультета присутсвует не число"; return; }

if (err == 3) { cout << "Недопустимый номер факультета"; return; }

cout << "Необработанная ошибка 11:" << err;

}

void err\_Training(int err) {

if (err == 1) { cout << "Недопустимое значени формы обучения"; return; }

if (err == 2) { cout << "Недопустимое значени формы обучения"; return; }

cout << "Необработанная ошибка 12:" << err;

}

void err\_Course(int err) {

if (err == 1) { cout << "Недопустимое значение курса"; return; }

if (err == 2) { cout << "В номере курса присутсвует не число"; return; }

if (err == 3) { cout << "Недопустимое значение курса"; return; }

cout << "Необработанная ошибка 13:" << err;

}

void err\_Group(int err) {

if (err == 1) { cout << "Недопустимая длинна значения номера группы"; return; }

if (err == 2) { cout << "В номере группа присутствует не число"; return; }

if (err == 3) { cout << "Недопустимое значение номера группы"; return; }

cout << "Необработанная ошибка 14:" << err;

}

void err\_Payment(int err) {

if (err == 1) { cout << "Недопустимое значение коммерцианализированности группы"; return; }

if (err == 2) { cout << "Недопустимое значение коммерцианализированности группы"; return; }

cout << "Необработанная ошибка 15:" << err;

}

void err\_Year(int err) {

if (err == 1) { cout << "Неправильная длинна значения года"; return; }

if (err == 2) { cout << "В значении года присутствует не число"; return; }

if (err == 3) { cout << "Недопустимое значение года"; return; }

cout << "Необработанная ошибка 16:" << err;

}

void err\_Department(int err) {

if (err == 1) { cout << "Недопустимая длинна значения номера кафедры"; return; }

if (err == 2) { cout << "В номере кафедры присутсвуют не числа"; return; }

if (err == 3) { cout << "Недопустимое значение номера кафедры"; return; }

cout << "Необработанная ошибка 17:" << err;

}

void err\_Date(int err) {

if (err == 1) { cout << "Недопустимое значение длинны даты"; return; }

if (err == 2) { cout << "Не корректная форма записи даты"; return; }

if (err == 3) { cout << "В значении даты присутсвуют не числа"; return; }

if (err == 4) { cout << "Недопустимое значение года"; return; }

if (err == 5) { cout << "Недопустимое зеначние месяца"; return; }

if (err == 6) { cout << "Недопустимое значение дня"; return; }

cout << "Необработанная ошибка 18:" << err;

}

void err\_Gender(int err) {

if (err == 1) { cout << "Недопустимая длинна значения пола"; return; }

if (err == 2) { cout << "Недопустимое значение пола"; return; }

cout << "Необработанная ошибка 19:" << err;

}

void err\_Address(int err) {

if (err == 1) { cout << "Отсутствует адрес"; return; }

if (err == 2) { cout << "В значении адреса присутсвуеют недопустимые символы"; return; }

cout << "Необработанная ошибка 20:" << err;

}

void err\_Status(int err) {

if (err == 1) { cout << "Недопустимая длинна значения статуса обучения"; return; }

if (err == 2) { cout << "В значении статуса обучения присутсвуют не числа"; return; }

if (err == 3) { cout << "Недопустимое значение статуса обучения"; return; }

cout << "Необработанная ошибка 21:" << err;

}

void err\_Subject(int err) {

if (err == 1) { cout << "Отсутсвует предмет"; return; }

if (err == 2) { cout << "В значении предмета присутсвуют недопустимые символы"; return; }

cout << "Необработанная ошибка 22:" << err;

}

void err\_TeacherSurname(int err) {

if (err == 1) { cout << "Фамилия преподователя начинается не с большой буквы"; return; }

if (err == 2) { cout << "В фамилии преподователя присутствуют недопустимые символы"; return; }

cout << "Необработанная ошибка 23:" << err;

}

void err\_TeacherName(int err) {

if (err == 1) { cout << "Отсутсвует имя преподователя"; return; }

if (err == 2) { cout << "Имя преподователя начинается не с большой буквы"; return; }

if (err == 3) { cout << "В имени преподователя присутствуют недопустимые символы"; return; }

cout << "Необработанная ошибка 24:" << err;

}

void err\_TeacherPatronymic(int err) {

if (err == 1) { cout << "Отчество преподователя начинается не с большой буквы"; return; }

if (err == 2) { cout << "В отчестве преподователя присутствуют недопустимые символы"; return; }

cout << "Необработанная ошибка 25:" << err;

}

void err\_Value(int err) {

if (err == 1) { cout << "В значении оценки присутсвует не число"; return; }

if (err == 2) { cout << "Недопустимое значение оценки"; return; }

if (err == 3) { cout << "Недопустимое значение оценки"; return; }

cout << "Необработанная ошибка 26:" << err;

}

void err\_Room(int err) {

if (err == 1) { cout << "Отсутствует номер ведомости"; return; }

if (err == 2) { cout << "В номере ведомости присутствует не число"; return; }

cout << "Необработанная ошибка 27:" << err;

}

void err\_ChecksumData(int err) {

if (err == 1) { cout << "Не совпадает контрольная сумма всех значений" << fileDam; return; }

cout << "Необработанная ошибка 28:" << err;

}

void err\_UniqueGroup(int err) {

if (err == 1) { cout << "У одинаковых групп разные данные, отличается форма обучения"; return; }

if (err == 2) { cout << "У одинаковых групп разные данные, отличается коммерцианализированность группы"; return; }

if (err == 3) { cout << "У одинаковых групп разные данные, отличается год поступления"; return; }

if (err == 4) { cout << "У одинаковых групп разные данные, отличается кафедра"; return; }

cout << "Необработанная ошибка 29:" << err;

}

void err\_UniqueNumber(int err) {

if (err == 1) { cout << "Пользователь с таким уникальным номером уже существует" << fileDam; return; }

cout << "Необработанная ошибка 30:" << err;

}

void err\_UniqueSNA(int err) {

if (err == 1) { cout << "В одной группе не допускается наличие студентов с одинаковами ФИО"; return; }

cout << "Необработанная ошибка 31:" << err;

}

void err\_UniqueRoom(int err) {

if (err == 1) { cout << "У одинаковых ведомостей разные данные, отличается дата"; return; }

if (err == 2) { cout << "У одинаковых ведомостей разные данные, отличается предмет"; return; }

if (err == 3) { cout << "У одинаковых ведомостей разные данные, отличается фамилия преподователя"; return; }

if (err == 4) { cout << "У одинаковых ведомостей разные данные, отличается имя преподователя"; return; }

if (err == 5) { cout << "У одинаковых ведомостей разные данные, отличается отчество преподователя"; return; }

cout << "Необработанная ошибка 32:" << err;

}

### Файл «Functions.h»

#ifndef CW\_FUNCTIONS\_H

#define CW\_FUNCTIONS\_H

#include <iostream> //Вывод

#include <fstream> //Файлы

#include <Windows.h> //Виндовс

#include <climits> //Константы

#include <iomanip> //Вы,вод

#include <string> //Строки

#include <conio.h> //"Консоль"

#include <ctime> //Время

#include <cstdio> //Файлы

#include "Struct.h"

#include "Line.h"

void test();

void readLine(string& strLine, LineNode\* line);

void readFileLine(istream& stream, InstituteNode& institute);

void readFromFile(InstituteNode& institute);

void writeNumber(ostream& stream, string\* number);

void writeStudentLine(ostream& stream, StudentNode\* node);

void writeRecordLine(ostream& stream, RecordNode\* node);

void writeFailLine(ostream& stream, LineNode\* node);

void writeToFile(InstituteNode& institute, int flag);

void createStydent(InstituteNode& institute);

void editStydent(InstituteNode& institute);

void deleteStydent(InstituteNode& institute);

void printStydent(InstituteNode& institute);

void createRecord(InstituteNode& institute);

void editRecord(InstituteNode& institute);

void deleteRecord(InstituteNode& institute);

void printRecord(InstituteNode& institute);

void deleteGroup(InstituteNode& institute);

void printGroup(InstituteNode& institute);

InstituteNode createInstitutesList();

#endif

### Файл «Functions.cpp»

#include "Functions.h"

using namespace std;

void test() {

cout << "------------------------------------" << endl;

//cout << INT\_MAX << endl;

//char c = 't';

//cout << c << endl;

//ofstream fout;

//fout.open("Проверка1.ins");

//fout << "123 Да Нет Yes No";

//fout.close();

//ifstream fin;

//fin.open("Проверка1.ins");

//string str1;

//getline(fin, str1);

//cout << str1 << endl;

cout << "------------------------------------" << endl;

}

void readLine(string& strLine, LineNode\* line) {

//line->coutStr();

checkLine checkLineFile\_Start[] = { checkLine\_Syntax, checkLine\_Type, checkLine\_AmountData }; //Старотовая проверка строки

checkLine checkLineFile\_Total[] = { checkLine\_Number, checkLine\_ChecksumNumber, checkLine\_Surname, checkLine\_Name, checkLine\_Patronymic, checkLine\_Institute, checkLine\_Faculty, checkLine\_Training , checkLine\_Course, checkLine\_Group, checkLine\_Payment, checkLine\_Year, checkLine\_Department, checkLine\_Date }; //Общая проверка файла

checkLine checkLineFile\_Student[] = { checkLine\_Gender, checkLine\_Address, checkLine\_Status }; //Проверка студента для файла

checkLine checkLineFile\_Record[] = { checkLine\_Subject, checkLine\_TeacherSurname, checkLine\_TeacherName, checkLine\_TeacherPatronymic, checkLine\_Value, checkLine\_Room }; //Проверка ведомости для файла

int size\_checkLineFile\_Start = sizeof(checkLineFile\_Start) / sizeof(checkLineFile\_Start[0]);

int size\_checkLineFile\_Total = sizeof(checkLineFile\_Total) / sizeof(checkLineFile\_Total[0]);

int size\_checkLineFile\_Student = sizeof(checkLineFile\_Student) / sizeof(checkLineFile\_Student[0]);

int size\_checkLineFile\_Record = sizeof(checkLineFile\_Record) / sizeof(checkLineFile\_Record[0]);

line->errorLine[0] = checkLine\_Lenght(strLine, line);

if (line->errorLine[0] == 0) {

for (int i = 0; i < size\_checkLineFile\_Start; i++) {

line->errorLine[i] = checkLineFile\_Start[i](strLine, line);

}

line->setErr();

if (line->getErr() == 0) {

for (int i = 4; i < (size\_checkLineFile\_Total + 4); i++) {

line->errorLine[i] = checkLineFile\_Total[i - 4](strLine, line);

}

if (line->type == 'S') {

for (int i = 0; i < size\_checkLineFile\_Student; i++) {

line->errorStudent[i] = checkLineFile\_Student[i](strLine, line);

}

}

if (line->type == 'R') {

for (int i = 0; i < size\_checkLineFile\_Record; i++) {

line->errorRecord[i] = checkLineFile\_Record[i](strLine, line);

}

}

line->errorLine[18] = checkLine\_ChecksumData(strLine, line);

}

}

line->setErr();

//line->coutError();

}

void readFileLine(istream& stream, InstituteNode& institute) {

string strLine = "";

getline(stream, strLine);

deleteSpace(strLine);

if (strLine == "") {

return;

}

LineNode\* line = new LineNode;

line->createStr(strLine);

readLine(strLine, line);

institute.add(line);

}

void readFromFile(InstituteNode& institute) {

cout << endl;

ifstream instituteDataFile;

instituteDataFile.open(institute.fileToWork);

if (!instituteDataFile) {

cout << "Ошибка чтения файла " << institute.fileToWork << endl;

return;

}

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

cout << "Ошибка открытия файла " << institute.fileToWork << endl;

return;

}

cout << "Открыт файл " << institute.fileToWork << endl;

istream\* stream = nullptr;

stream = &instituteDataFile;

while (!stream->eof()) {

readFileLine(\*stream, institute);

}

instituteDataFile.close();

cout << "Чтение файла " << institute.fileToWork << " закончено" << endl;

if (institute.nodeSuccess != 0) {

cout << "Успешно загружено " << institute.nodeSuccess << " " << institute.worlEnd\_запись[getWordEnd(institute.nodeSuccess)] << endl;

if (institute.nodeMistake != 0) {

cout << "Не удалось загрузить " << institute.nodeMistake << " " << institute.worlEnd\_запись[getWordEnd(institute.nodeMistake)] << endl;

cout << "Список не загруженных записей:" << endl;

cout << endl;

}

else {

cout << "Не загруженных записей нет" << endl;

}

}

else {

cout << "Файл пуст" << endl;

}

institute.resetSM();

institute.printFailList(1);

}

void writeNumber(ostream& stream, string\* number) {

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

stream << number[i][j];

}

if (i != 7) {

stream << "-";

}

}

}

void writeStudentLine(ostream& stream, StudentNode\* node) {

stream << "[S]:";

writeNumber(stream, node->number);

node->writeGeneral(stream, node);

stream << node->gender << ";" << node->address << ";" << node->status;

stream << ";" << node->checksumData;

stream << ";";

stream << endl;

}

void writeRecordLine(ostream& stream, RecordNode\* node) {

stream << "[R]:";

writeNumber(stream, node->number);

node->writeGeneral(stream, node);

stream << node->subject << ";" << node->teacherSurname << ";" << node->teacherName << ";" << node->teacherPatronymic << ";" << node->value << ";" << node->room;

stream << ";" << node->checksumData;

stream << ";";

stream << endl;

}

void writeFailLine(ostream& stream, LineNode\* node) {

stream << node->getStr();

stream << endl;

}

void writeToFile(InstituteNode& institute, int flag) {

cout << endl;

ofstream instituteDataFile;

instituteDataFile.open(institute.fileToWork);

if (!instituteDataFile) {

cout << "Ошибка создания файла " << institute.fileToWork << endl;

return;

}

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

cout << "Ошибка открытия файла " << institute.fileToWork << endl;

return;

}

cout << "Создан файл " << institute.fileToWork << endl;

ostream\* stream = nullptr;

stream = &instituteDataFile;

StudentNode\* nodeS = institute.headStudentsList;

while (nodeS != nullptr) {

writeStudentLine(\*stream, nodeS);

institute.nodeSuccess++;

nodeS = nodeS->nextNode();

}

instituteDataFile << endl;

RecordNode\* nodeR = institute.headRecordsList;

while (nodeR != nullptr) {

writeRecordLine(\*stream, nodeR);

institute.nodeSuccess++;

nodeR = nodeR->nextNode();

}

if (flag) {

instituteDataFile << endl;

LineNode\* nodeF = institute.failList;

while (nodeF != nullptr) {

writeFailLine(\*stream, nodeF);

institute.nodeMistake++;

nodeF = nodeF->nextNode(nodeF);

}

}

instituteDataFile.close();

cout << "Зпись файла " << institute.fileToWork << " закончена" << endl;

cout << "Успешно загружено " << institute.nodeSuccess << " " << institute.worlEnd\_запись[getWordEnd(institute.nodeSuccess)];

if (institute.nodeMistake != 0) {

cout << " и " << institute.nodeMistake << " не загрузившихся " << institute.worlEnd\_запись[getWordEnd(institute.nodeMistake)];

}

cout << endl;

institute.resetSM();

}

void createStydent(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->fillStydent(line);

if (outFlag) {

cout << endl;

cout << "Создание нового студента отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

line->errorUnique[0] = institute.checkUniqueGroup(line);

if (line->errorUnique[0] == 0) {

line->errorUnique[2] = institute.checkUniqueSNP(line);

if (line->errorUnique[2] == 0) { //Само добавление

cout << endl;

cout << "Добавление нового студента..." << endl;

institute.generateNumber(line);

line->checksumNumber = getChecksum(getNumber(line->number));

string data = "";

data = data + line->type + getNumber(line->number) + line->checksumNumber + line->surname + line->name + line->patronymic + line->institute + line->faculty + line->training + line->course + line->group + line->payment + line->year + line->department + line->date;

data = data + line->gender + line->address + line->status;

line->checksumData = getChecksum(data);

institute.add(line);

institute.resetSM();

cout << "Новый студент успешно добавлен!" << endl;

return;

}

else { //errU 2

line->printErr(line);

cout << "Студент не создан" << endl;

}

}

else { //errU 0

line->printErr(line);

cout << "Студент не создан" << endl;

}

}

void editStydent(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findStydent(line);

if (outFlag) {

cout << endl;

cout << "Редактирование студента отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск студента..." << endl;

StudentNode\* node = institute.findStudent(line);

if (line != nullptr) {

delete line;

}

if(node == nullptr) {

cout << endl;

cout << "Студент не найден" << endl;

return;

}

else { //Само редактирование

node->printStudentCard();

cout << endl;

char key;

bool run = 1;

cout << "Продолжить редактирование?" << endl;

cout << "1 Да" << endl;

cout << "2 Нет" << endl;

cout << endl;

while (run) {

key = \_getch();

switch (key) {

case kESC:

case k2:

system("CLS");

cout << "Редактирование остановлено" << endl;

run = 0;

break;

case k1:

case kENTER: { //Редактирование

LineNode\* line = node->copyStudent();

system("CLS");

int outFlag = line->fillStydent(line, 1);

if (outFlag) {

cout << endl;

cout << "Редактирование студента отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

line->errorUnique[0] = institute.checkUniqueGroup(line);

if (line->errorUnique[0] == 0) {

if ((line->institute == node->institute) && (line->faculty == node->faculty) && (line->course == node->course) && (line->group == node->group)) {

line->errorUnique[2] = 0;

}

else {

line->errorUnique[2] = institute.checkUniqueSNP(line);

}

if (line->errorUnique[2] == 0) {

cout << endl;

cout << "Внесение изменений..." << endl;

string data = "";

data = data + line->type + getNumber(line->number) + line->checksumNumber + line->surname + line->name + line->patronymic + line->institute + line->faculty + line->training + line->course + line->group + line->payment + line->year + line->department + line->date;

data = data + line->gender + line->address + line->status;

line->checksumData = getChecksum(data);

institute.deleteOneStydent(node->number);

institute.add(line);

institute.resetSM();

cout << "Студент успешно отредактирован!" << endl;

return;

}

else { //errU 2

line->printErr(line);

cout << endl;

cout << "Студент не отредактирован" << endl;

}

}

else { //errU 0

line->printErr(line);

cout << endl;

cout << "Студент не отредактирован" << endl;

}

run = 0;

break;

} //Редактирование

default:

break;

}

}

}

}

void deleteStydent(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findStydent(line);

if (outFlag) {

cout << "Удаление студента отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск студента..." << endl;

StudentNode\* node = institute.findStudent(line);

if (line != nullptr) {

delete line;

}

if (node == nullptr) {

cout << endl;

cout << "Студент не найден" << endl;

return;

}

else { //Само удаление

node->printStudentCard();

cout << endl;

char key;

bool run = 1;

cout << "Продолжить удаление?" << endl;

cout << "1 Да" << endl;

cout << "2 Нет" << endl;

cout << endl;

while (run) {

key = \_getch();

switch (key) {

case kESC:

case k2:

system("CLS");

cout << "Удаление остановлено" << endl;

run = 0;

break;

case kENTER:

case k1:

institute.deleteOneStydent(node);

cout << endl;

cout << "Студент удалён" << endl;

run = 0;

break;

default:

break;

}

}

}

}

void printStydent(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findStydent(line);

if (outFlag) {

cout << endl;

cout << "Поиск студента отменён" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск студента..." << endl;

StudentNode\* node = institute.findStudent(line);

if (line != nullptr) {

delete line;

}

if (node == nullptr) {

cout << endl;

cout << "Указанный студент не найден" << endl;

return;

}

else {

node->printStudentCard();

RecordNode\* record = institute.headRecordsList;

cout << endl;

while (record != nullptr) {

if ((institute.compareNumber(node->number, record->number) == 1) && (record->value != "-")) {

institute.operationSuccess++;

record->printStudentOne();

}

record = record->nextNode();

}

if (institute.operationSuccess == 0) {

cout << "На данного студента не найдены ведомости" << endl;

}

institute.resetSM();

}

}

void createRecord(InstituteNode& institute) {

LineNode\* line = new LineNode; //Сначала поиск группы

system("CLS");

int outFlag = line->findGroup(line);

if (outFlag) {

cout << endl;

cout << "Поиск группы отменён" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск группы..." << endl;

StudentNode\* node = institute.findGroup(line);

if (node == nullptr) {

cout << endl;

cout << "Указанная группа не найдена" << endl;

return;

}

else { //Если группа найдена, то продолжаем создавать ведомость

system("CLS");

int outFlag = line->fillRecord(line);

if (outFlag) {

cout << "Создание новой ведомости отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << "Создание новой ведомости..." << endl;

line->training = node->training;

line->payment = node->payment;

line->year = node->year;

line->department = node->department;

line->room = institute.generateRecordRoom();

StudentNode\* node = institute.headStudentsList;

while (node != nullptr) {

if ((node->institute == line->institute) && (node->faculty == line->faculty) && (node->course == line->course) && (node->group == line->group)) {

LineNode\* temp = line->copyLineNode();

copyNumber(node->number, temp->number);

temp->checksumNumber = node->checksumNumber;

temp->surname = node->surname;

temp->name = node->name;

temp->patronymic = node->patronymic;

string data = "";

data = data + temp->type + getNumber(temp->number) + temp->checksumNumber + temp->surname + temp->name + temp->patronymic + temp->institute + temp->faculty + temp->training + temp->course + temp->group + temp->payment + temp->year + temp->department + temp->date;

data = data + temp->subject + temp->teacherSurname + temp->teacherName + temp->teacherPatronymic + temp->value + temp->room;

temp->checksumData = getChecksum(data);

institute.add(temp);

institute.resetSM();

} //if

node = node->nextNode();

} //while

cout << "Новая ведомость успешно создана!" << endl;

cout << "Уникальный номер:" << line->room << endl;

if (node != nullptr) { delete node; }

if (line != nullptr) { delete line; }

return;

} //else

}

void editRecord(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findRecord(line);

if (outFlag) {

cout << endl;

cout << "Редактирование ведомости отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск ведомости..." << endl;

RecordNode\* record = institute.findRecord(line);

if (line != nullptr) {

delete line;

}

if (record == nullptr) {

cout << endl;

cout << "Указанная ведомость не найдена" << endl;

return;

}

else {

record->printHeadRecordCard();

char key;

bool run = 1;

cout << endl;

cout << "Продолжить редактирование?" << endl;

cout << "1 Да" << endl;

cout << "2 Нет" << endl;

cout << endl;

while (run) {

key = \_getch();

switch (key) {

case kESC:

case k2:

system("CLS");

cout << "Редактирование остановлено" << endl;

run = 0;

break;

case kENTER:

case k1: {

system("CLS");

record->printHeadRecordCard();

record->printHeadRecordCardSNP();

RecordNode\* node = institute.headRecordsList;

while (node != nullptr) { //while для прохода по всем ведомостям и печати полей этой ведомости

if (node->room == record->room) {

institute.operationSuccess++;

node->printEditRecordOne(institute.operationSuccess);

}

node = node->nextNode();

}

node = institute.headRecordsList;

LineNode\* line = new LineNode;

while (node != nullptr) { //while для прохода по всем ведомостям и редактирования оценок

if (node->room == record->room) {

institute.operationMistake++;

line->value = node->value;

outFlag = node->editRecord(line, institute.operationSuccess, institute.operationMistake);

if (outFlag) {

cout << endl;

cout << "Редактирование ведомости отменено, все уже внесённые изменения сохранены" << endl;

if (line != nullptr) { delete line; }

institute.resetSM();

return;

}

node->value = line->value;

string data = ""; // Подсчёт новой контрольной суммы данных

data = data + "R" + getNumber(node->number) + node->checksumNumber + node->surname + node->name + node->patronymic + node->institute + node->faculty + node->training + node->course + node->group + node->payment + node->year + node->department + node->date;

data = data + node->subject + node->teacherSurname + node->teacherName + node->teacherPatronymic + node->value + node->room;

node->checksumData = getChecksum(data);

}

node = node->nextNode();

}

institute.resetSM();

if (line != nullptr) { delete line; }

cout << endl;

cout << "Изменения в ведомость успешно внесены!" << endl;

run = 0;

break;

}

default:

break;

}

} //while

}

}

void deleteRecord(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findRecord(line);

if (outFlag) {

cout << endl;

cout << "Удаление ведомости отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск ведомости..." << endl;

RecordNode\* node = institute.findRecord(line);

if (line != nullptr) {

delete line;

}

if (node == nullptr) {

cout << endl;

cout << "Указанная ведомость не найдена" << endl;

return;

}

else {

node->printHeadRecordCard();

char key;

bool run = 1;

cout << endl;

cout << "Продолжить удаление?" << endl;

cout << "1 Да" << endl;

cout << "2 Нет" << endl;

cout << endl;

while (run) {

key = \_getch();

switch (key) {

case kESC:

case k2:

system("CLS");

cout << "Удаление остановлено" << endl;

run = 0;

break;

case kENTER:

case k1: {

LineNode\* line = node->copyRecord();

institute.deleteRecord(line);

cout << endl;

cout << "Ведомость удалена" << endl;

if (line != nullptr) {

delete line;

}

run = 0;

break;

}

default:

break;

}

} //while

}

}

void printRecord(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findRecord(line);

if (outFlag) {

cout << endl;

cout << "Поиск ведомости отменён" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск ведомости..." << endl;

RecordNode\* node = institute.findRecord(line);

if (line != nullptr) {

delete line;

}

if (node == nullptr) {

cout << endl;

cout << "Указанная ведомость не найдена" << endl;

return;

}

else {

node->printHeadRecordCard();

node->printHeadRecordCardSNP();

RecordNode\* record = institute.headRecordsList;

while (record != nullptr) {

if (node->room == record->room) {

if (getValue(node->value) != "") {

institute.operationSuccess++;

record->printRecordOne(institute.operationSuccess);

}

}

record = record->nextNode();

}

institute.resetSM();

}

}

void deleteGroup(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findGroup(line);

if (outFlag) {

cout << endl;

cout << "Удаление группы отменено" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск группы..." << endl;

StudentNode\* node = institute.findGroup(line);

if (line != nullptr) {

delete line;

}

if (node == nullptr) {

cout << endl;

cout << "Указанная группа не найдена" << endl;

return;

}

else {

node->printHeadGroupCard();

char key;

bool run = 1;

cout << endl;

cout << "Продолжить удаление?" << endl;

cout << "1 Да" << endl;

cout << "2 Нет" << endl;

cout << endl;

while (run) {

key = \_getch();

switch (key) {

case kESC:

case k2:

system("CLS");

cout << "Удаление остановлено" << endl;

run = 0;

break;

case kENTER:

case k1: {

LineNode\* line = node->copyStudent();

institute.deleteGroup(line);

cout << endl;

cout << "Группа удалена" << endl;

if (line != nullptr) {

delete line;

}

run = 0;

break;

}

default:

break;

}

} //while

}

}

void printGroup(InstituteNode& institute) {

LineNode\* line = new LineNode;

system("CLS");

int outFlag = line->findGroup(line);

if (outFlag) {

cout << endl;

cout << "Поиск группы отменён" << endl;

if (line != nullptr) {

delete line;

}

return;

}

cout << endl;

cout << "Поиск группы..." << endl;

StudentNode\* node = institute.findGroup(line);

if (line != nullptr) {

delete line;

}

if (node == nullptr) {

cout << endl;

cout << "Указанная группа не найдена" << endl;

return;

}

else {

node->printHeadGroupCard();

StudentNode\* student = institute.headStudentsList;

while (student != nullptr) {

if ((node->institute == student->institute) && (node->faculty == student->faculty) && (node->course == student->course) && (node->group == student->group)) {

institute.operationSuccess++;

student->printStudentOne(institute.operationSuccess);

}

student = student->nextNode();

}

if (student != nullptr) { delete student; }

institute.resetSM();

}

}

InstituteNode createInstitutesList() {

InstituteNode institute;

institute.fileToWork = institute.path + institute.file + institute.expansion;

institute.fileTheName = institute.path + institute.file + institute.expansion;

readFromFile(institute);

institute.printLineStudents();

institute.printLineRecords();

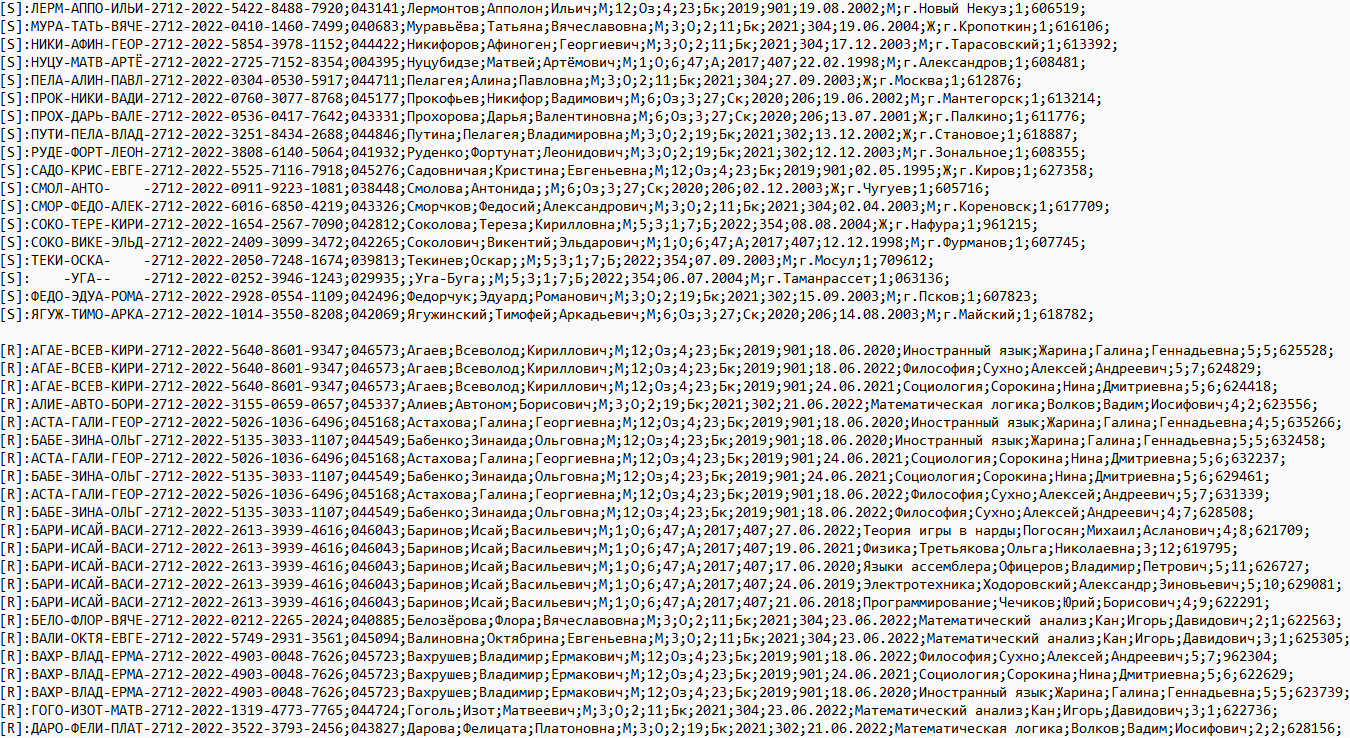
return institute;

}

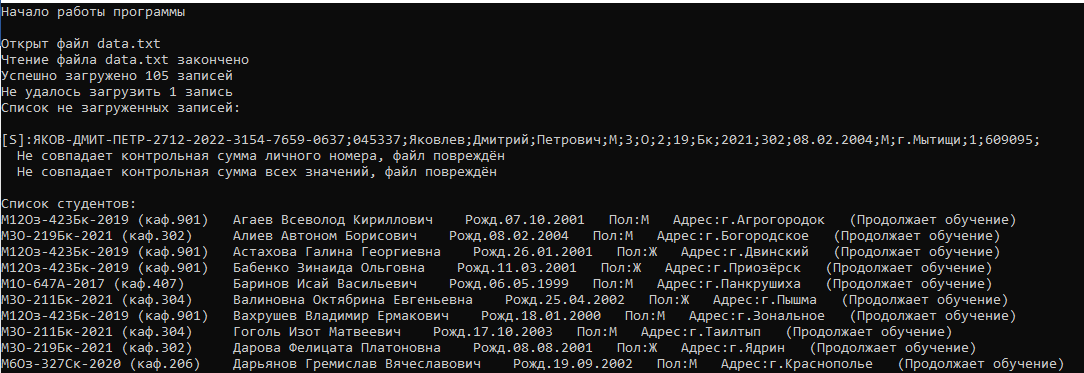
# Результат работы программы

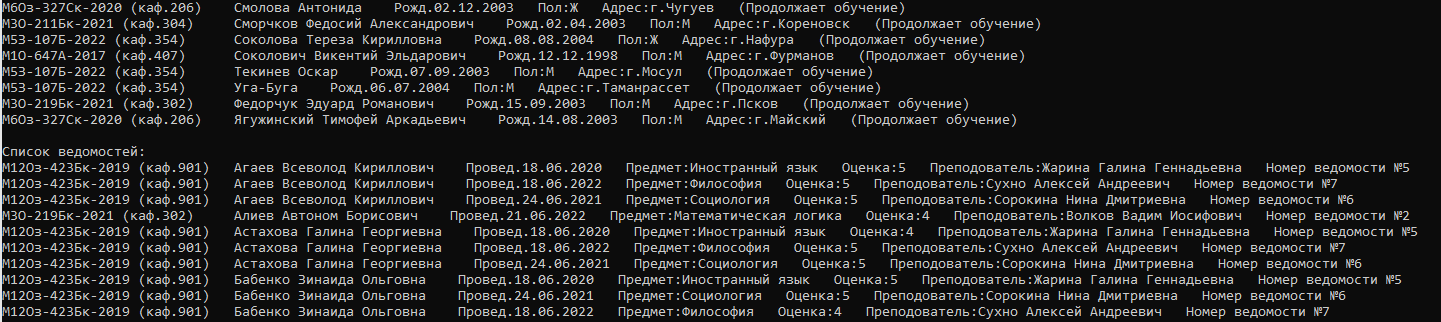
### Запуск и хранение данных

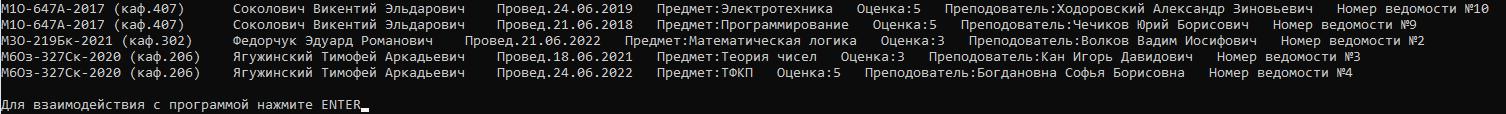
При запуске программы она автоматически открывает и считывает файл data.txt, размещённый в одной директории с программой.

Пример файла data.txt

После запуска программа выводит не обработанные линии, а так же список всех записей студентов и каждый отдельной записи оценки в ведомость:

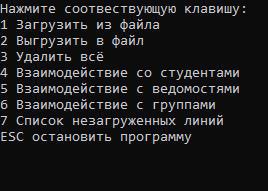




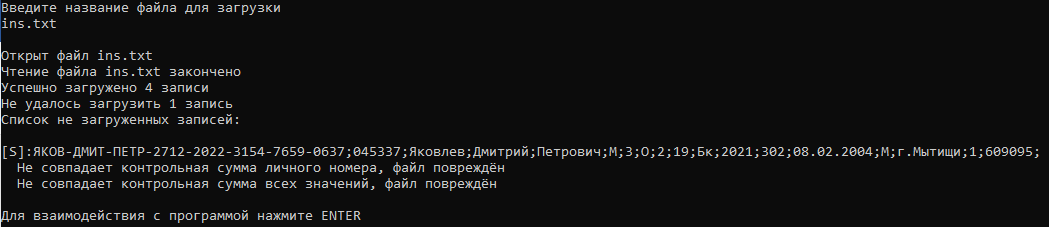


### Меню

Для взаимодействия с программой реализовано меню. После запуска программы и считывания файла пользователь видит меню, и по нажатию на нужную кнопку клавиатуры выбирает соответствующий пункт:

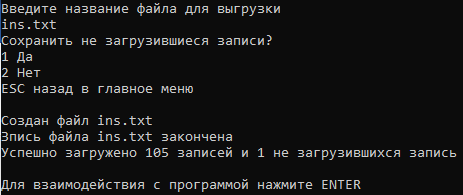


### 1 Загрузить из файла

После выбора данного пункта пользователь должен написать название файла, из которого он хочет дополнить данные. После этого заданный файл считается, выведутся не считанные строки и причина ошибки:

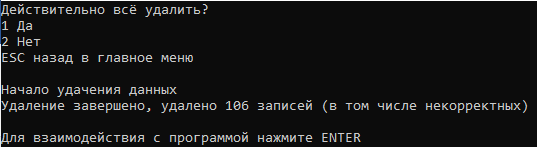
### 2 Выгрузить в файл

После выбора данного пункта пользователь должен написать название файла, в который он хочет сохранить данные. После этого создаться заданный файл и туда сохранятся сначала все записи о студентах, затем о ведомостях. Перед записью пользователя спросят нужно ли сохранить ранее не обработанные строки:



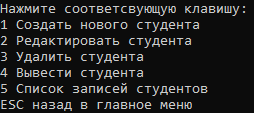
### 3 Удалить всё

При выборе данного пункта пользователь может удалить все данные, загруженные в программу. Перед этим у пользователя уточнят о выборе:

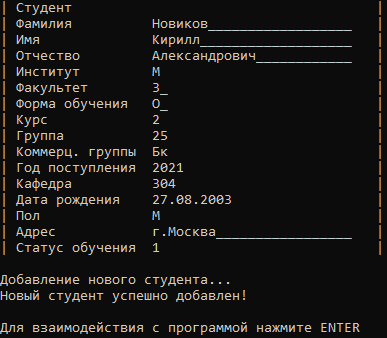


### 4 Взаимодействие со студентами

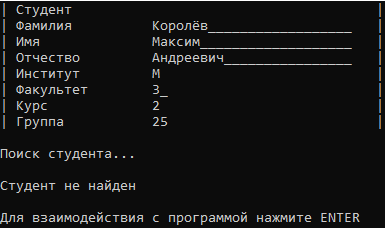
При выборе данного меню пользователю откроется подменю с данными пунктами:



1 Создать нового студента: пользователю нужно будет заполнить карточку нового студента, заполнив все поля. Поля «Форма обучения», «Коммерц. группы» , «Год поступления» и «Кафедра» при условии существования группы, в которую планируется добавить студента, должный ей совпадать, иначе выдаст ошибку. Цифрам статуса обучения соответствует 0 – «Отчислен», 1 – «Продолжает обучение», 2 – «В академическом отпуске», 3 – «Закончил обучение». При прохождении всех проверок студент будет добавлен в список.

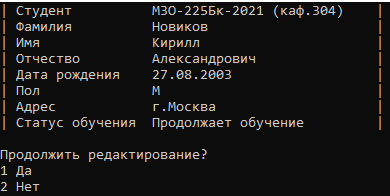


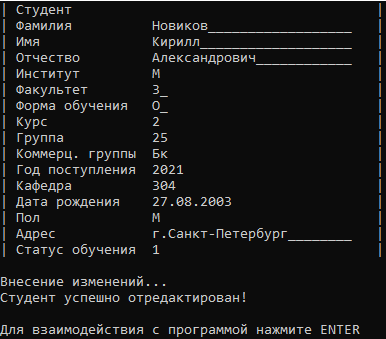
2 Редактировать студента: пользователь должен будет найти студента, которого желает отредактировать, путём заполнения полей с данными. Если он не будет найден программа об этом сообщит.



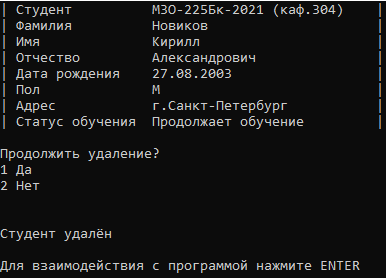
(пример отсутствия заданного студента)

В случаен нахождения студента у пользователя уточнят: тот ли студент. После его можно будет отредактировать.

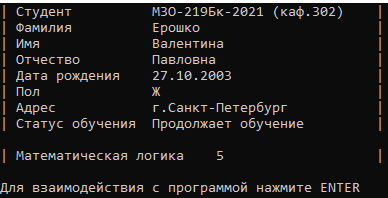


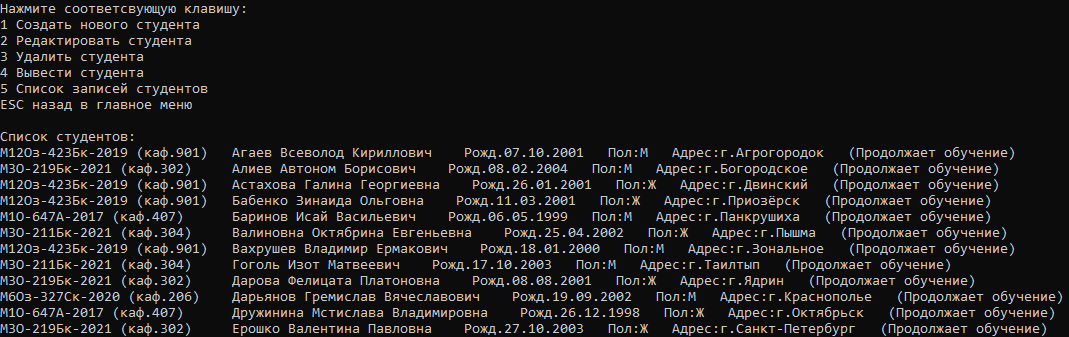


3 Удалить студента: после поиска студента аналогично п.2, пользователь сможет удалить студента:

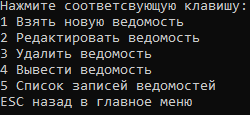


4 Вывести студента: после поиска студента аналогично п.2, пользователю покажут подробную информацию о студенте и список оценок при наличии:

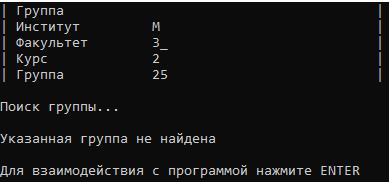


5 Список записей студентов: будет напечатан список всех загруженных студентов:

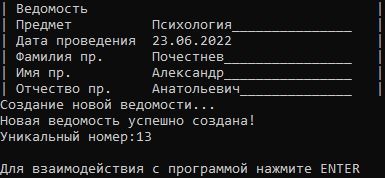
### 5 Взаимодействие с ведомостями

При выборе данного меню пользователю откроется подменю с данными пунктами:

1 Взять новую ведомость: пользователь должен будет заполнить поля поиска группы:

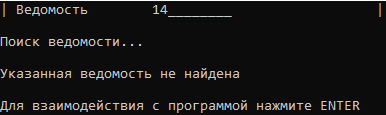


В случае нахождения данной группы в списке пользователь сможет заполнить поля ведомости:

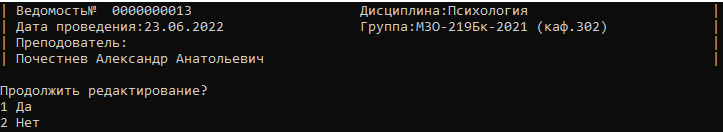


Пользователю напишет уникальный номер, присвоенный данной ведомости. Если он его забыл, то данную ведомость с номером можно будет найти в пункте «5 Список записей ведомостей», который будет рассмотрен далее.

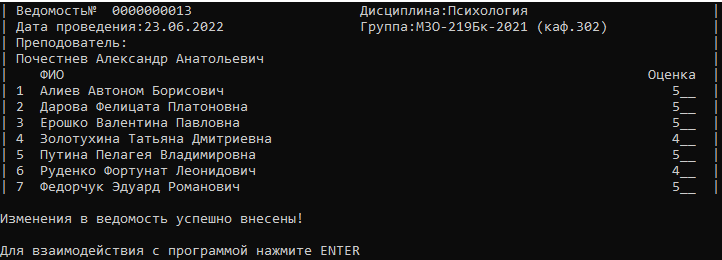
2 Редактировать ведомость: пользователь должен будет указать уникальный номер ведомости и, после дополнительно уточнения, заполнить оценки в ведомости. Оценки можно менять неограниченное количество раз:



(пример отсутствия ведомости с данным уникальным номером)

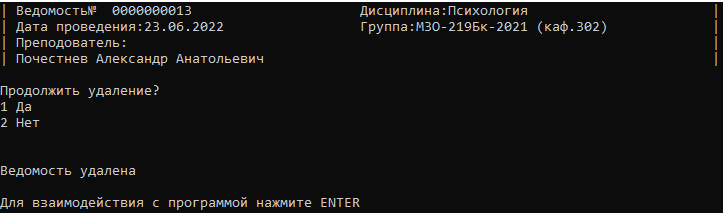


(после ввода «13»)

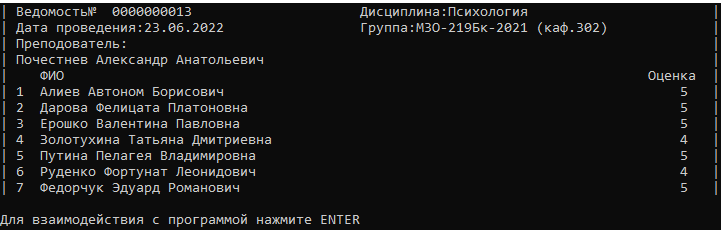


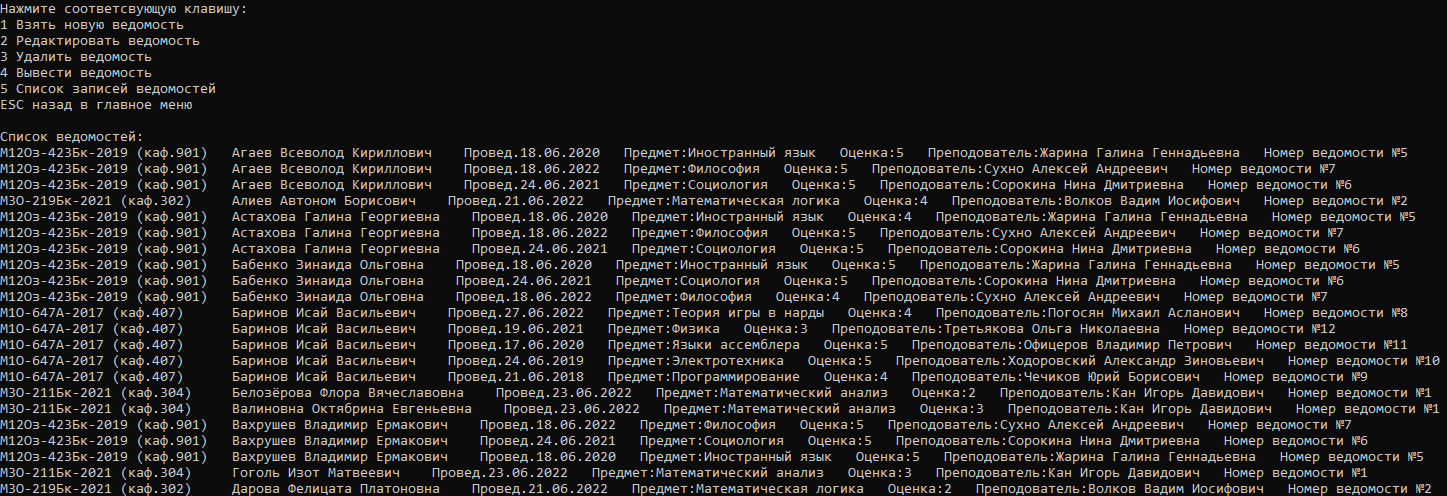
(после заполнения полей оценок в ведомости)

3 Удалить ведомость: после поиска ведомости аналогично п.2, будет предложено удалить ведомость:



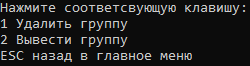
4 Вывести ведомость: после поиска ведомости аналогично п.2, будет напечатана ведомость с оценками:



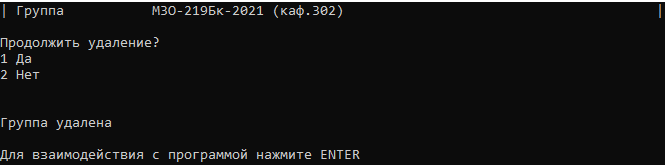
5 Список записей ведомостей: будет напечатан список всех записей ведомостей:

### 6 Взаимодействие с группами

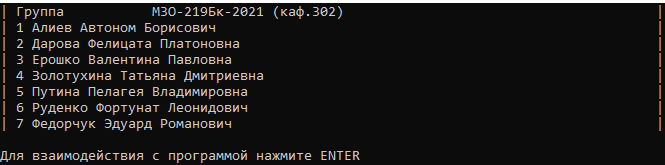
При выборе данного меню пользователю откроется подменю с данными пунктами:



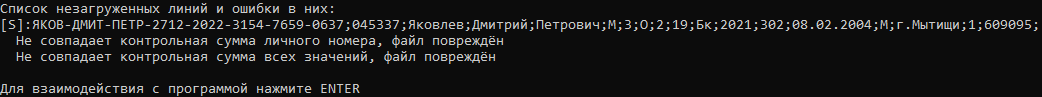
1 Удалить группу: после поиска группы (аналогично «5 Взаимодействие с ведомостями» -> «1 Взять новую ведомость»), после соглашения пользователя, будут удалены все студенты, принадлежащей данной группе:



2 Вывести группу: после поиска группы (аналогично «5 Взаимодействие с ведомостями» -> «1 Взять новую ведомость»), будет напечатан список группы:



### 7 Список не загруженных линий

После выбора данного пункта будут напечатаны линии, не загруженные из файла, и почему он были не загружены:

При нажатии клавиши ESC будет осуществлён выход из программы. При её нажатии во время поиска/изменения данных будет осуществлён выход в меню.

# **Вывод**:

В результате выполнения курсовой работы была разработана система управления БД студентов, их групп и оценок на ЯП C++. Для хранения данных используется текстовый файл, который защищён от редактирования контрольными суммами каждой строки. Управления записями осуществляется через использование меню. После чтения файла данные записываются в двунаправленный список в программе. Разработка программы происходила с среде разработки Visual Studio.