Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Рязанский государственный радиотехнический университет им. В.Ф. Уткина

Кафедра электронных вычислительных машин

|  |  |
| --- | --- |
|  | **К ЗАЩИТЕ**  руководитель КП  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.И. Елесина  «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2023 г. |

**КУРСОВАЯ РАБОТА**

по дисциплине

**«Основы алгоритмизации и ООП»**

на тему

**«Разработка приложения**

**с использованием динамических структур данных»**

Выполнил студент группы 240

Балоян А.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата сдачи на проверку, подпись

Руководитель проекта

к.т.н., доцент Елесина С.И. \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

оценка дата защиты, подпись

Рязань 2023 г.

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**Рязанский государственный радиотехнический университет имени В.Ф. Уткина**

**Кафедра ЭВМ**

**ЗАДАНИЕ**

**на курсовую работу**

**по дисциплине «Основы алгоритмизации и ООП»**

Студенту **Балояну А.А., гр. 240**

**1. Тема работы:** «Разработка приложения с использованием динамических структур

данных»

**2. Срок сдачи студентом законченной работы** 30 мая 2023 г.

**3. Руководитель работы:** Елесина Светлана Ивановна, к.т.н., доцент, доцент кафедры ЭВМ

РГРТУ

**4. Исходные данные к работе**

1. *Операционная система Windows XP/7/8/10*

2. *Язык* *программирования* *С/С++*

3. *Среда программирования MS Visual Studio C++*

4. *Индивидуальное задание*

**5. Содержание пояснительной записки**

*Задание*

*Содержание*

*Введение*

1. *Постановка задачи*

2. *Разработка алгоритмов*

3. *Разработка программы*

4. *Экспериментальная проверка программы*

5. *Руководство оператора.*

*Заключение.*

*Список использованных источников.*

*Приложение – листинги программ.*

Задание выдано «20» марта 2023 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / доцент каф. ЭВМ С.И. Елесина

**Индивидуальное задание к курсовой работе**

**по дисциплине «Основы алгоритмизации и ООП»**

***студенту группы 240 Балояну А.А.***

**Общие требования**

Вариант задания связан с разработкой таблицы данных с использованием линейных

однонаправленных списков.

Разрабатываемая программа должна обязательно выполнять следующие запросы:

− заполнение пустой таблицы;

− сохранение таблицы в файле;

− чтение таблицы из файла;

− вывод таблицы на экран;

− добавление элементов в таблицу;

− удаление элементов из таблицы;

− редактирование элементов таблицы;

− а также все запросы, которые указаны в индивидуальном задании.

Вызовы запросов должны осуществляться через систему меню с использованием

средств визуального программирования Visual Studio (Visual C++). Необходимо

предусмотреть контроль ошибок пользователя при вводе данных. Результаты некоторых

запросов должны выводиться в виде графиков или диаграмм.

При запуске приложение должно выдавать заставку, которая отражает назначение

приложения.

Все элементарные действия должны быть оформлены в виде подпрограмм, а

некоторые объявления и подпрограммы должны быть оформлены в виде модуля (модулей).

**Индивидуальное задание**

**Вариант № 01. Абоненты библиотеки**

Информация об абонентах библиотеки следующая:

– номер читательского билета;

– ФИО;

– год рождения;

– пол;

– подразделение (кафедра, номер группы);

– должность;

– отметка о перерегистрации;

– имеются книги на срок;

– дата возврата книг.

Написать программу, которая выполняет следующие запросы:

– вывод информации об абоненте по номеру читательского билета;

– упорядочение таблицы по ФИО;

– вывод списка абонентов-должников определенного факультета;

– вывод списка абонентов, которые не прошли перерегистрацию;

– вывод процентного соотношения сотрудников и студентов среди абонентов.

Задание выдано «20» марта 2023 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / доцент каф. ЭВМ С.И. Елесина

**Содержание**

Оглавление

[Введение 5](#_Toc137585394)

[1. Постановка задачи 6](#_Toc137585395)

[2. Разработка алгоритмов 7](#_Toc137585396)

[3. Разработка программы 17](#_Toc137585397)

[4. Экспериментальная проверка программы 18](#_Toc137585398)

[5. Руководство оператора 22](#_Toc137585399)

[5.1 Назначение программы 22](#_Toc137585400)

[5.2 Условия выполнения программы 22](#_Toc137585401)

[5.3 Выполнение программы 22](#_Toc137585402)

[5.4 Сообщение оператору 23](#_Toc137585403)

[Заключение 25](#_Toc137585404)

[Список используемых источников 26](#_Toc137585405)

[Приложение А 27](#_Toc137585406)

# Введение

Данная курсовая работа посвящена разработке приложения с использованием динамических структур данных в контексте дисциплины "Основы алгоритмизации и ООП". Она имеет целью изучить принципы проектирования и реализации приложений, основанных на динамических структурах данных, и продемонстрировать их применение на практике.

В работе будут рассмотрены основные концепции алгоритмизации и объектно-ориентированного программирования, которые необходимы для разработки эффективных и удобных в использовании приложений. Будут изучены различные типы динамических структур данных, такие как списки, стеки, очереди, деревья и графы, а также алгоритмы их обработки. Конкретно в данной курсовой работе будет использоваться класс “Списки”.

Класс "Список" в C++ представляет собой структуру данных, которая позволяет хранить и организовывать элементы в линейном порядке. Он предоставляет гибкую динамическую структуру для хранения элементов различных типов данных.

Основные характеристики класса "Список" включают:

∙Динамическое добавление и удаление элементов

∙Обращение к элементам по индексу

∙Удобные операции вставки и удаления

∙Итерация по элементам списка

∙Различные методы доступа и модификации

В целом, класс "Список" в C++ предоставляет удобный интерфейс для работы с динамической структурой данных, позволяя эффективно управлять элементами списка и обрабатывать их в программе.

В ходе разработки также будет использоваться метод многомодульного программирования. Многомодульное программирование в C++ представляет собой подход к организации программного проекта, в котором код разбивается на отдельные модули или файлы для улучшения модульности, повторного использования и облегчения сопровождения программы.

Основные принципы многомодульного программирования в C++:

∙Разделение кода на модули

∙Использование заголовочных файлов

∙Компиляция модулей

∙Использование модульной системы и пространств имен

# 1. Постановка задачи

Целью курсовой работы являются разработка и отладка приложения,

написанного на алгоритмическом языке программирования C/C++ в

среде визуального программирования Visual C++.

Разрабатываемая программа «Абоненты библиотеки» должна будет выполнять следующие функции:

– Вывод информации об абоненте по номеру читательского билета;

– Упорядочение таблицы по ФИО

– Вывод списка абонентов-должников определённого факультета;

– Вывод списка абонентов, которые не прошли перерегистрацию;

– Вывод процентного соотношения сотрудников и студентов среди абонентов;

– Запись информации о каждом абоненте библиотеки в пустую таблицу;

– Сохранение информации о сотрудниках в файл;

– Редактирование информации о каждом абоненте;

– Удаление информации о любом из абонентов;

– Добавление информации о новом абоненте библиотеки в существующую таблицу.

Для разработки графического интерфейса будут использоваться элементы Windows Forms: DataGridView, TextBox, ComboBox, Button и т.д.



Рисунок 1 – Макет приложения.

# 2. Разработка алгоритмов



Рисунок 1 – Алгоритм вывода информации о k-том элементе

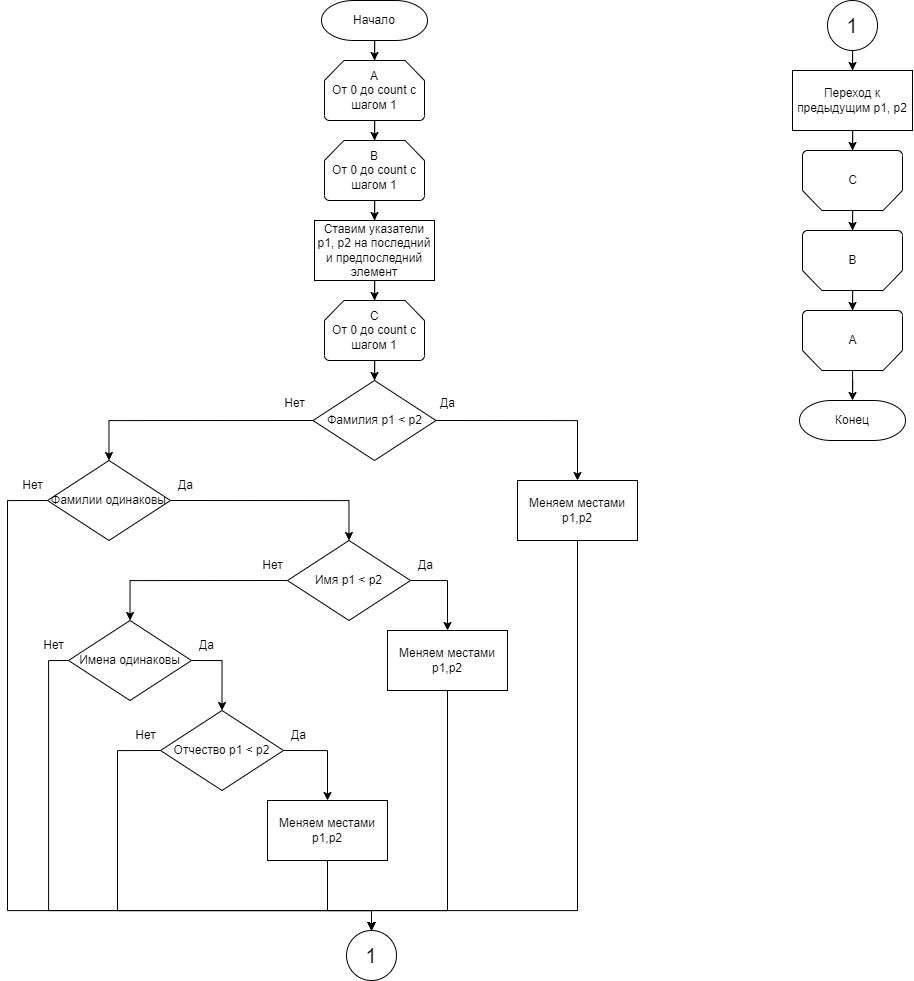


Рисунок 2 – Алгоритм сортировки по ФИО

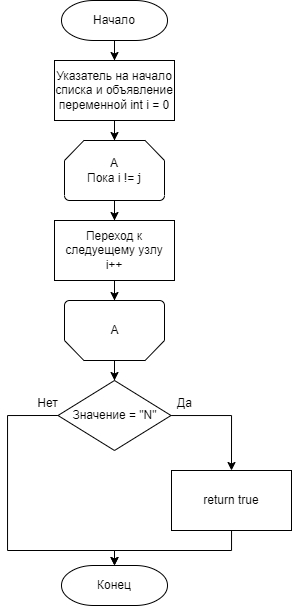


Рисунок 3 – Алгоритм поиска не прошедших перерегистрацию

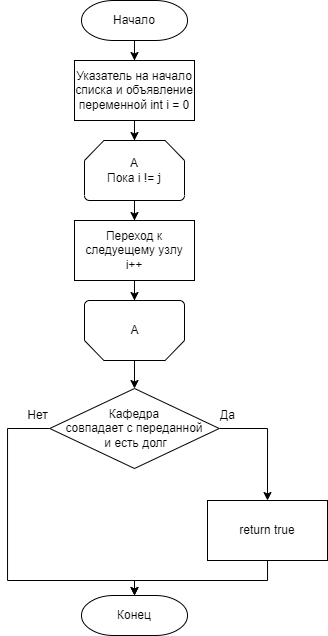


Рисунок 4 – Алгоритм нахождения должника по факультету



Рисунок 5 – Алгоритм вычисления количества сотрудников

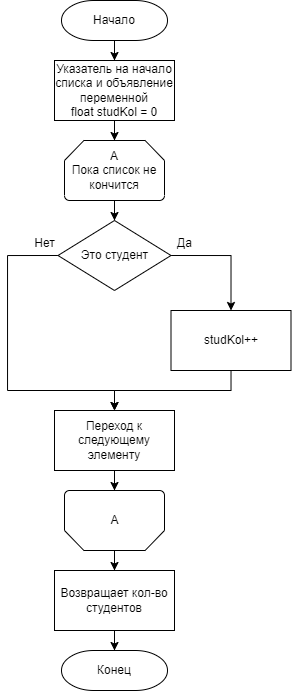


Рисунок 6 – Алгоритм вычисления количества студентов

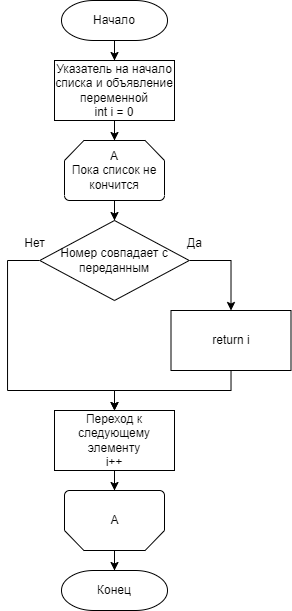


Рисунок 7 – Алгоритм поиска по номеру

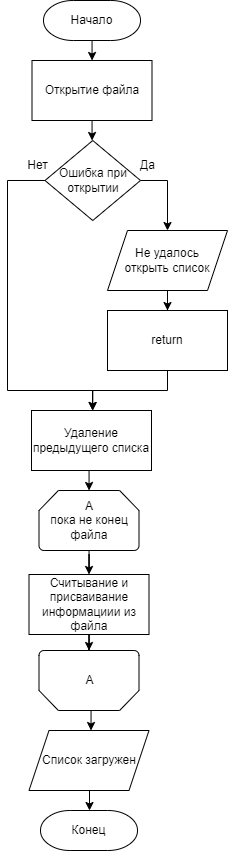


Рисунок 8 –Алгоритм чтения из файла



Рисунок 9 – Алгоритм удаления списка

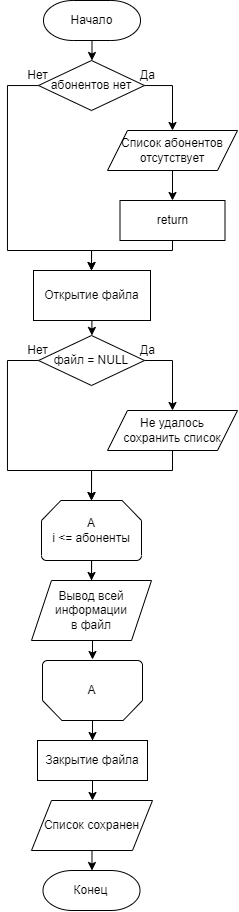


Рисунок 10 – Алгоритм сохранения в файл

# 3. Разработка программы

Программа разрабатывалась на языке программирования С++. Он широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования.

Данный язык программирования предоставляет нам в использование множество инструментов для комфортного и эффективного программирования. Конкретно в данной программе будут использоваться структуры и классы.  
 Также С++ даёт возможность разработки в среде CLR(Common Language Runtime), а в частности Windows Forms, с помощью которого был разработан пользовательский интерфейс.

Windows Forms предоставляет в распоряжение разработчика широкий выбор элементов для построения графического интерфейса, а также их гибкую настройку. В этой среде можно воплотить любые идеи.

При разработке программа была разбита на модули для удобства разработки. То есть был использован метод многомодульной разработки.

- Структура ABONENT содержит в себе поля: ФИО, номер билета, кафедра, группа, пол, год рождения, отметка о перерегистрации, должность, дата возврата книги, имеются книги на срок.

- Структура Node содержит в себе поля информации и указателя.

- Класс List поля: корень списка, кол-во узлов списка, переход к предыдущему узлу. Класс также содержит методы:

∙ List() – Конструктор класса;

∙ int getCount() - Получение количества узлов списка;

∙ bool isEmpty() - Проверка, пуст ли список;

∙ ABONENT getValue(Node\* p) - Получение значения узла списка;

∙ void setValue(Node\* p, int val) - Установка значения узла списка;

∙ Node\* getFirst() - Получение корневого узла списка;

∙ Node\* getLast() - Получение последнего узла списка;

∙ void Clear() - Очистка списка;

∙ Node\* Next(Node\*) - Переход к следующему узлу;

∙ Node\* Add(ABONENT abonent, Node\*) - Добавление узла списка;

∙ Node\* Delete(Node\*) - Удаление узла списка;

∙ ABONENT Print(int a) - Вывод значений узлов списка;

∙ void Swap(Node\*, Node\*) - Взаимообмен двух узлов;

∙ void SortByName() - Сортировка по имени;

∙ bool SortByPerereg(int j) - Сортировка по непрошедшим перерегистрацию;

∙ bool SortByDolg(string kafedra, int j) - Сортировка по должникам кафедры;

∙ float Sotrud() - Кол-во сотрудников;

∙ float Stud() - Кол-во студентов;

∙ int SeekByNum(int number) - Поиск по номеру билета;

∙ void WriteFile() - Сохранение в файл;

∙ void Readfile(char\* z) - Чтение из файла;

# 4. Экспериментальная проверка программы

Проведём экспериментальную проверку программы на корректную работу. Проверим исполнение функций данных в задании.

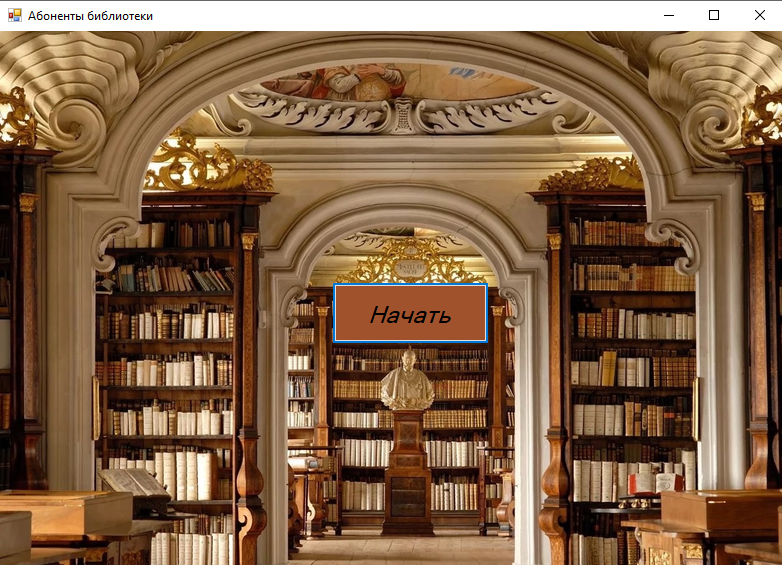


Рисунок 11 – Начальный экран программы

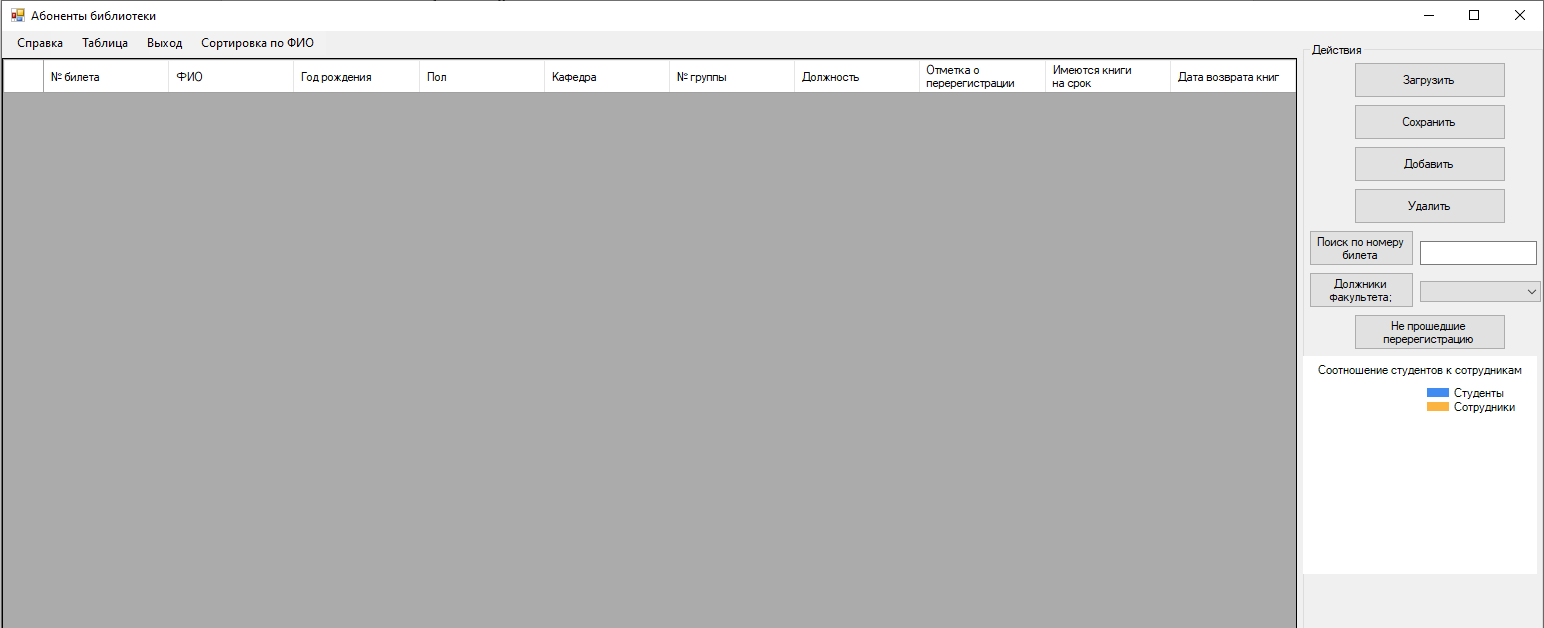


Рисунок 12 – Таблица после запуска программы

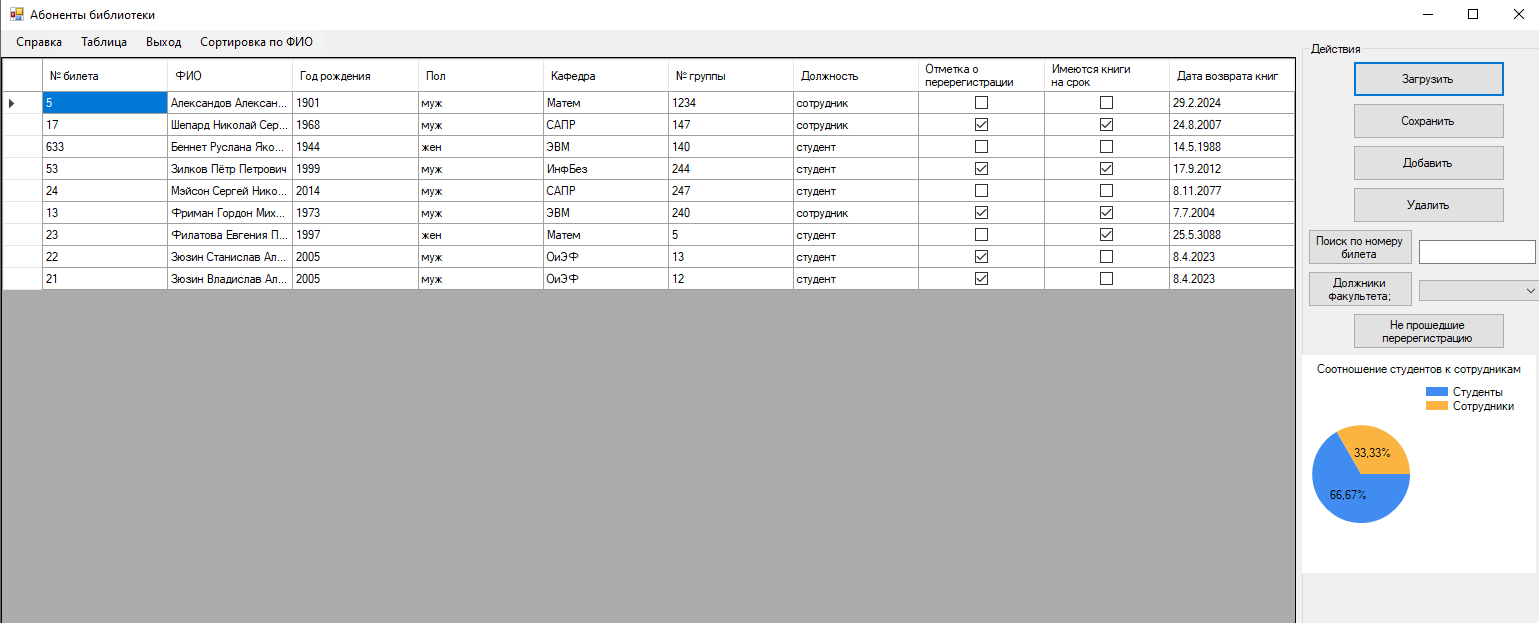


Рисунок 13 – Заполненная таблица из файла “абоненты.txt”

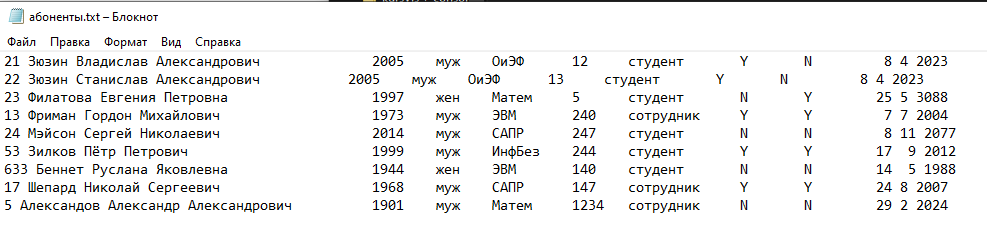


Рисунок 14 – Содержимое файла “абоненты.txt”

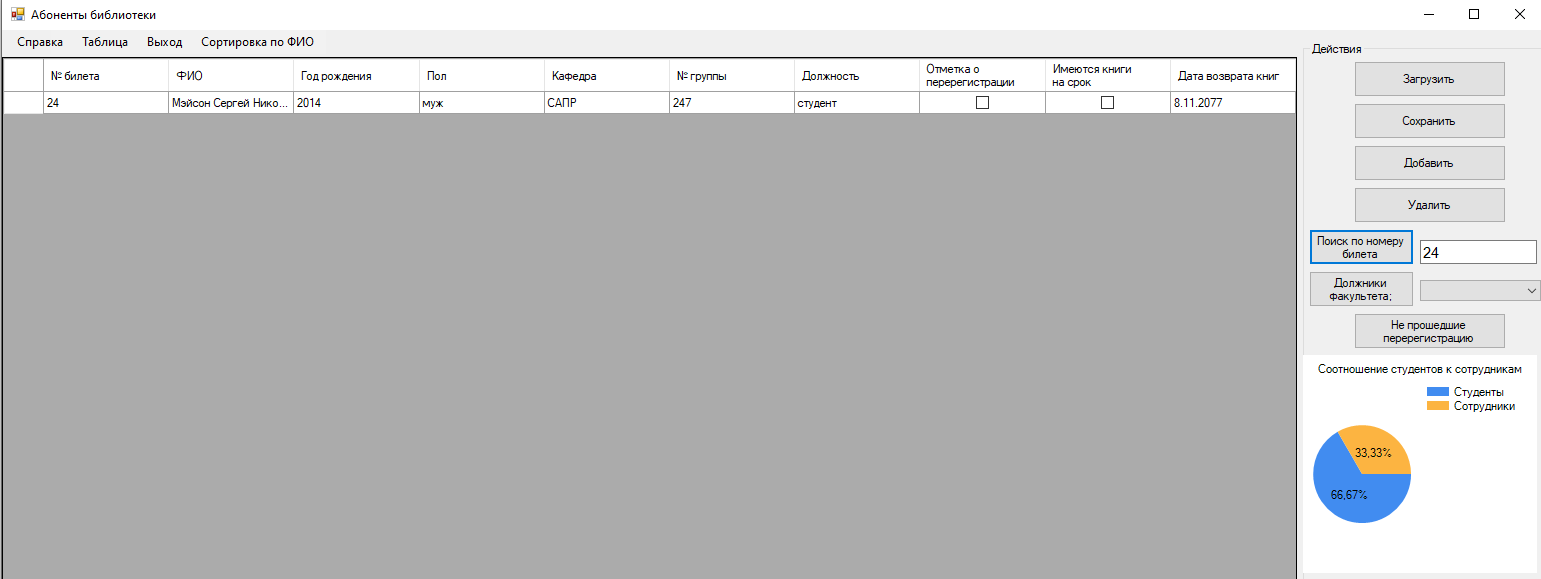


Рисунок 15 – Результат поиска по номеру

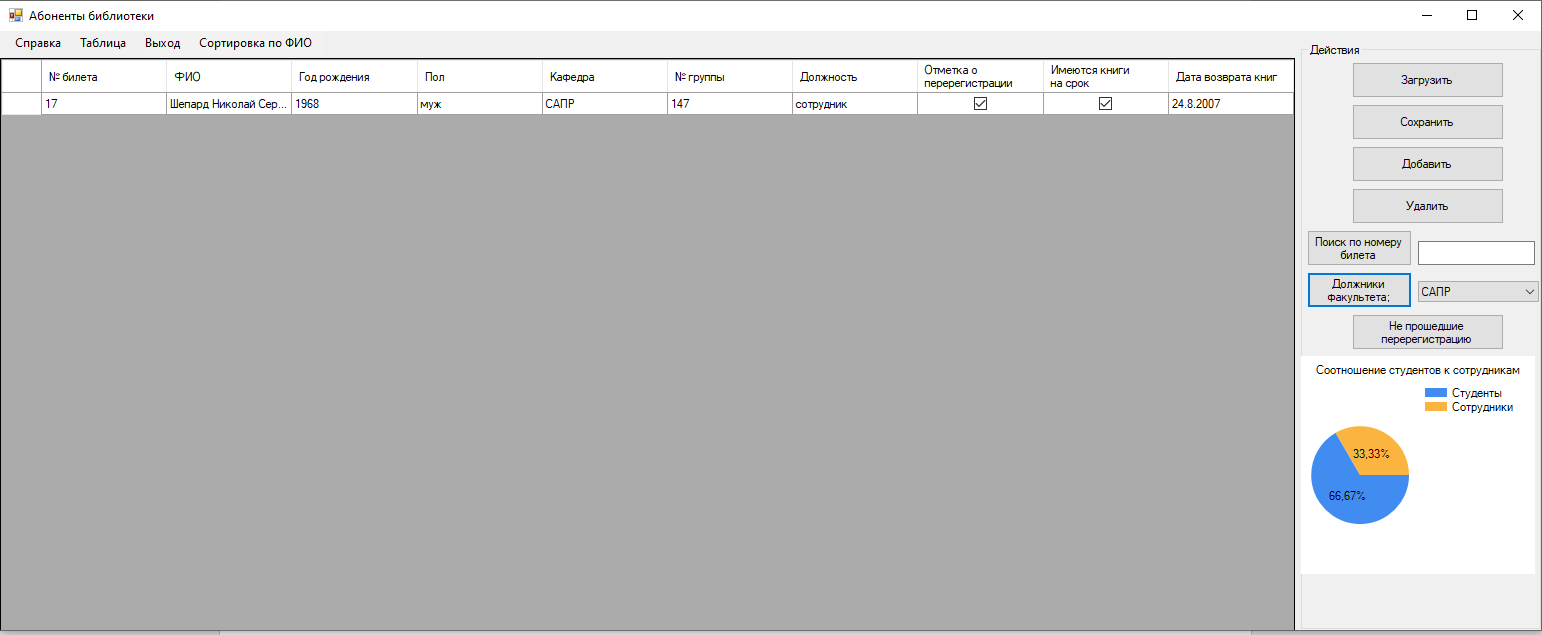


Рисунок 16 – Результат сортировки по должникам кафедры

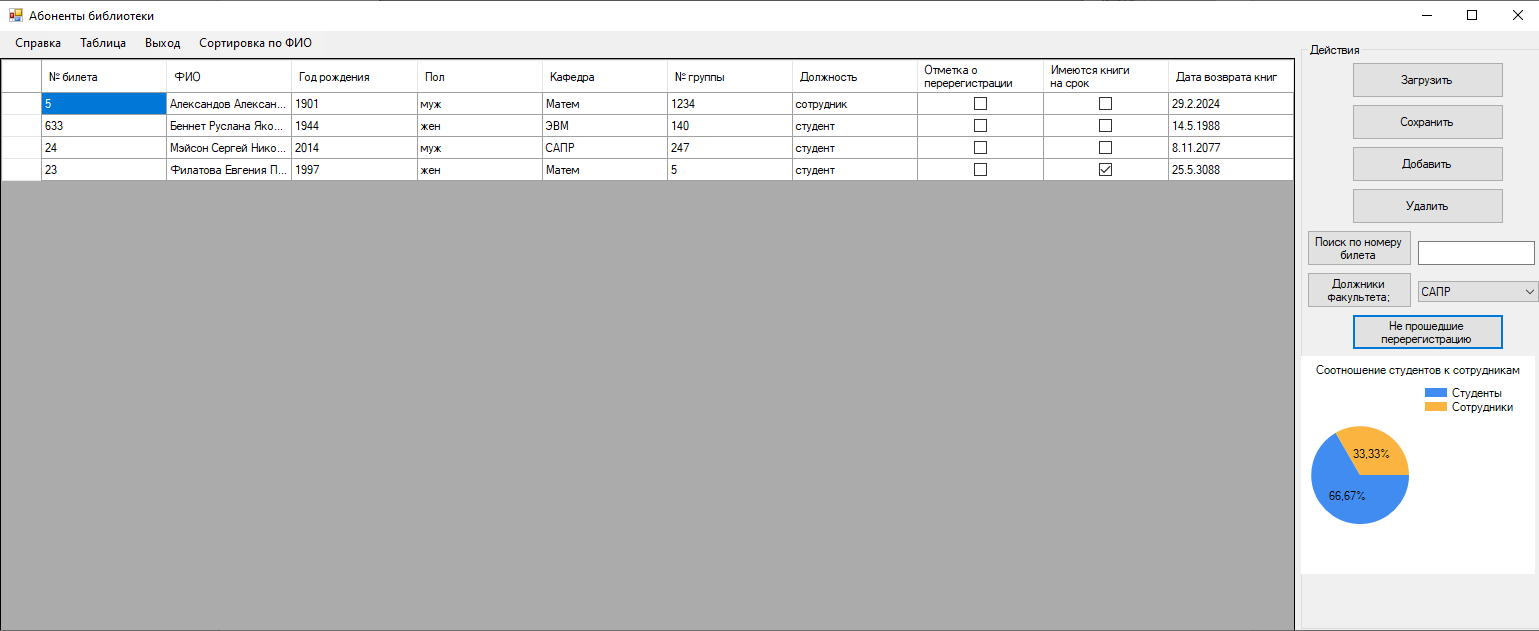


Рисунок 17 – Результат сортировки по непрошедшим перерегистрацию

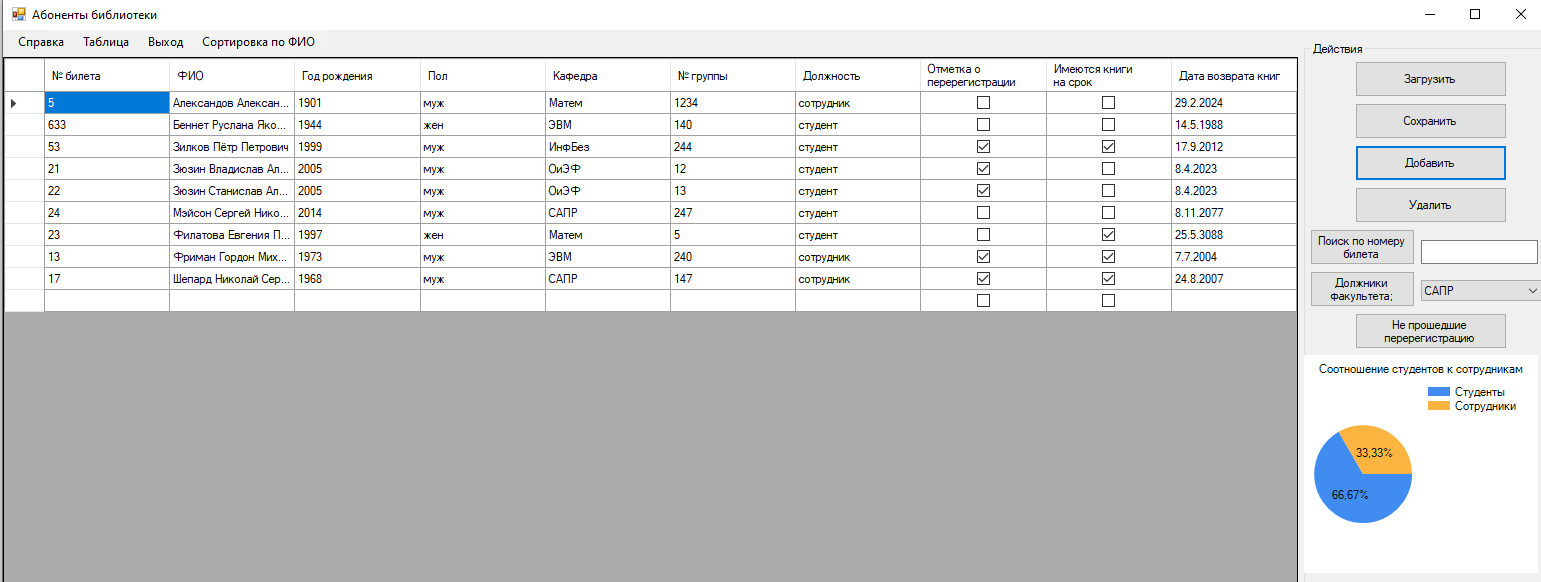


Рисунок 18 – Результат добавления нового абонента

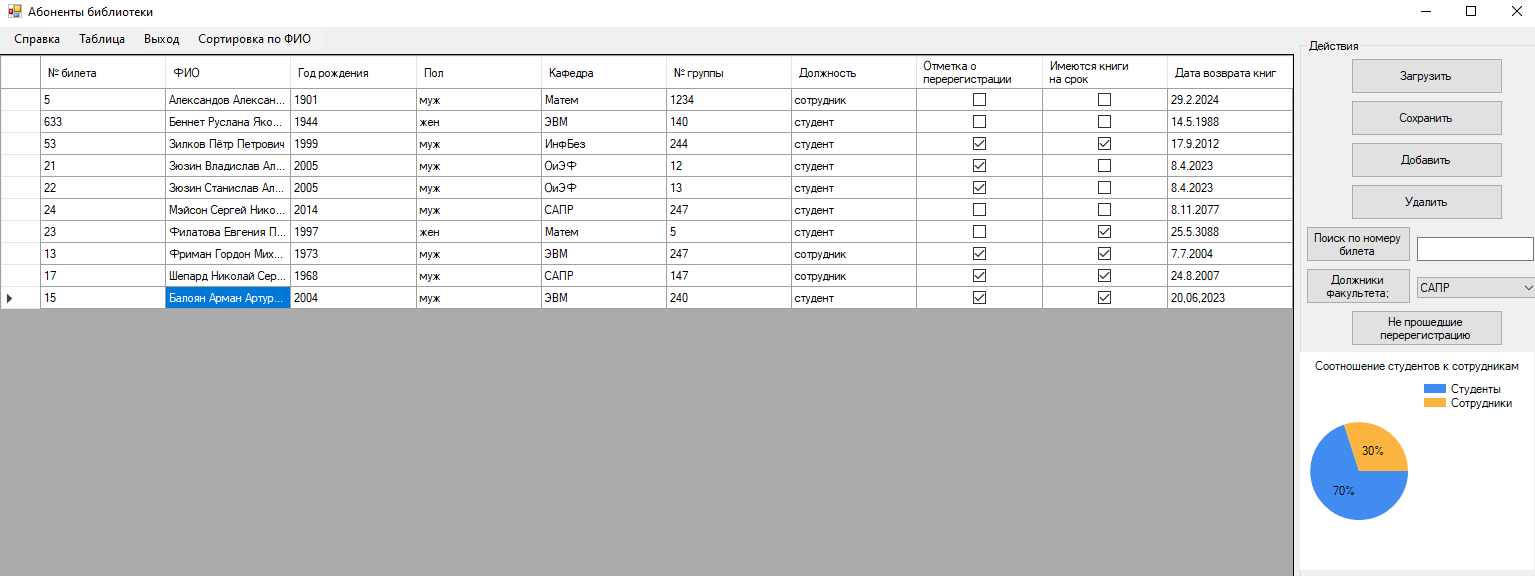


Рисунок 19 – Результат заполнения новой строки

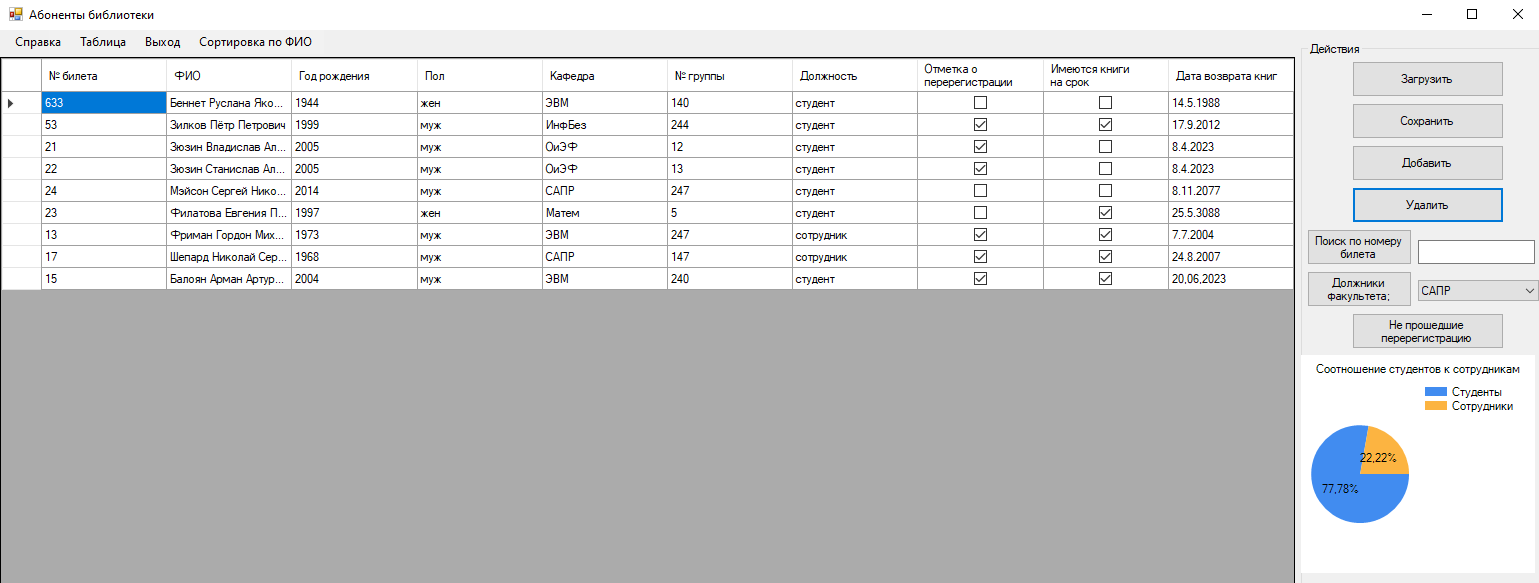


Рисунок 20 – Результат удаления первой строки

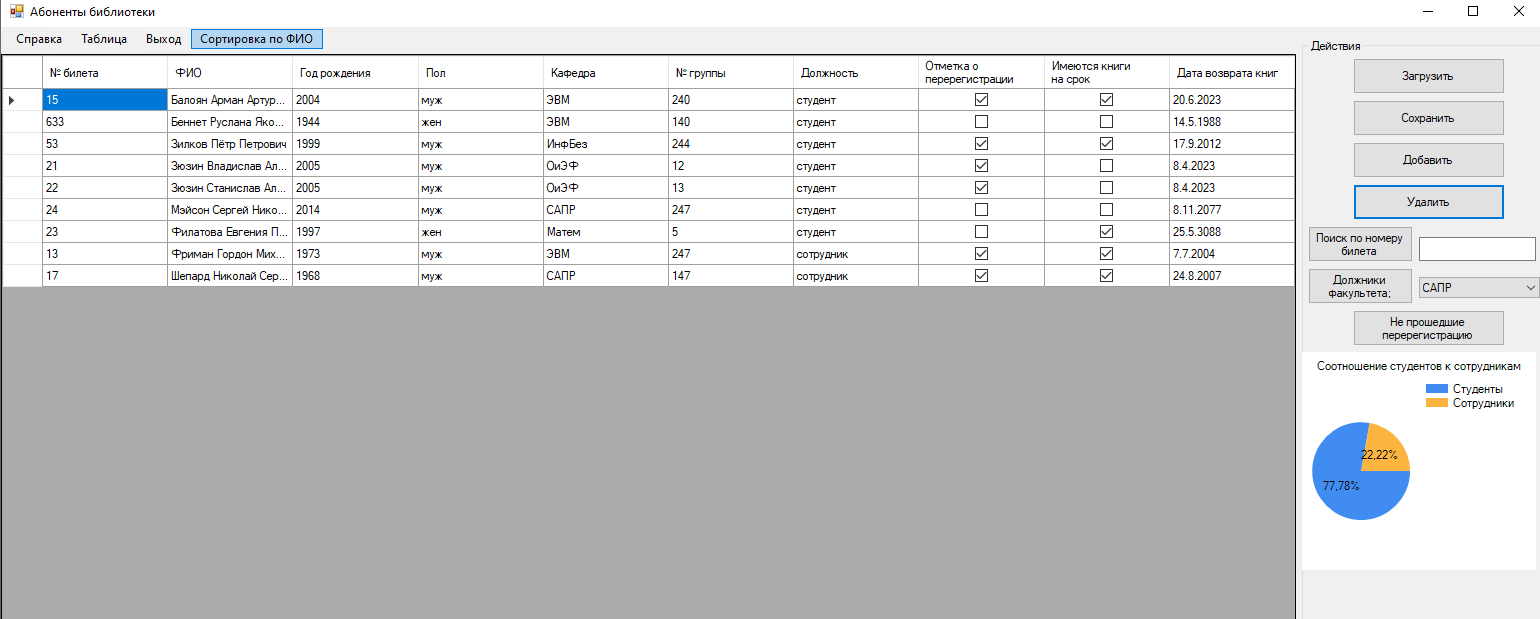


Рисунок 21 – Результат сортировки по ФИО

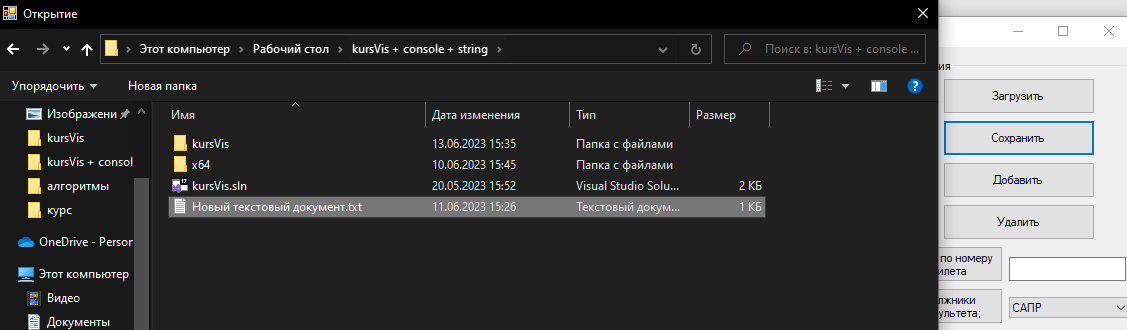


Рисунок 22 – Сохранение таблицы в файл

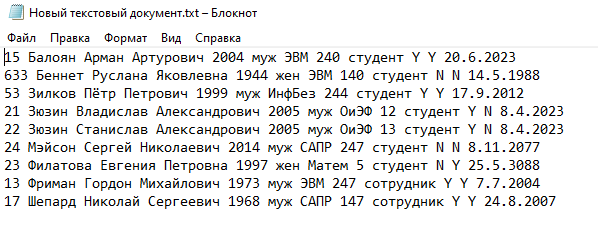


Рисунок 23 – Результат сохранения таблицы в файл

# 5. Руководство оператора

## 5.1 Назначение программы

Программа предназначена для работы со списком абонентов библиотеки. Она содержит в себе следующий функционал: работа с файлами (чтение и запись), сортировка по ФИО, поиск абонентов по номеру, фильтрация по непрошедшим перерегистрацию, по должникам определённой кафедры, удаление и добавление абонентов, также есть диаграмма, показывающая соотношение сотрудников библиотеки и студентов.

## 5.2 Условия выполнения программы

Для корректной и удобной работы с программой “Абоненты библиотеки” необходимо иметь в распоряжении ПК со следующими требованиями:

∙ Оперативная память минимум – 128 Мб;

∙ ОС Windows Xp/7/10

∙ Видеокарта с минимум – 128 Мб;

∙ Процессор с минимум 2-мя ядрами и 4-мя потоками;

∙ Свободное место – 256 Мб;

∙ Мышь;

∙ Клавиатура;

∙ Монитор;

## 5.3 Выполнение программы

Чтобы начать работу с программой, нужно сначала запустить “Library.exe”. После запуска появиться начальное окно программы, на котором нужно нажать кнопку “Начать”.

После этого появиться пустая таблица. Чтобы добавить в неё абонентов, вы можете либо нажать кнопку “Добавить”, которая добавит в таблицу пустую строку, требующую ручного заполнения, либо нажать на “Загрузить” и у вас появиться окно проводника для выбора файла откуда загрузится информация для заполнения таблицы. Для сохранения таблицы в файл нажмите на кнопку “Сохранить”, далее проводиться аналогичная процедуре загрузки в таблицу операция.

Если какая-то строка оказалась лишней или неактуальной, её можно выделить и нажать на кнопку “Удалить”, чтобы избавиться от неё. Для редактирования просто два раза щёлкните по ячейке, которую хотите редактировать.

Для сортировки абонентов по алфавиту нажмите на кнопку “Сортировка по ФИО”. Для поиска по номеру билета введите искомый номер в строку напротив кнопки “Поиск по номеру билета” и нажмите на ранее упомянутую кнопку. Для сортировки по должникам кафедры нажмите на кнопку напротив кнопки “Должники факультета”, оттуда раскроется список факультетов, среди которых выберите любой по вашему желанию, далее нажмите на кнопку “Должники факультета”. Для сортировки по непрошедшим перерегистрацию нажмите на одноимённую кнопку.

После некоторых сортировок вам может понадобиться вернуть таблицу в исходное состояние. Для этого нажмите на кнопку “Таблица”. Для выхода из программы есть кнопка “Выход”. Также есть кнопка “Справка” кратко описывающая суть программы.

## 5.4 Сообщение оператору

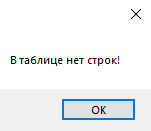


Рисунок 24 – Сообщение об ошибке при работе с пустой таблицей

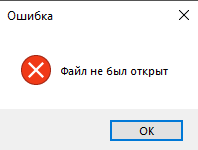


Рисунок 25 – Сообщение об ошибке при открытии файла

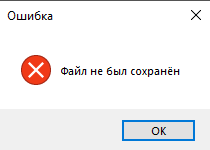


Рисунок 26 – Сообщение об ошибке при сохранении файла

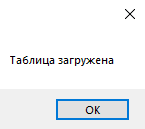


Рисунок 27 – Сообщение об успешной загрузке из файла

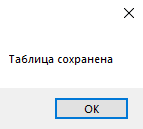


Рисунок 28 – Сообщение об успешном сохранении в файл

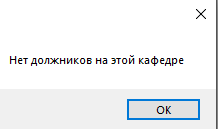


Рисунок 29 – Сообщение об отсутствии должников на кафедре

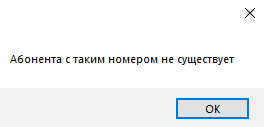


Рисунок 30 – Сообщение об отсутствии абонента с таким номером

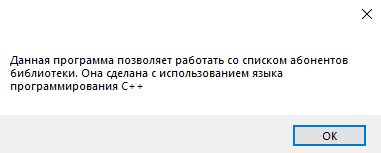


Рисунок 31 – Вывод справки о программе

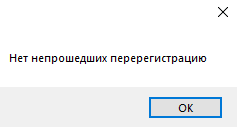


Рисунок 32 – Сообщение об отсутствии абонентов непрошедших перерегистрацию

# Заключение

В ходе данной курсовой работы были получены навыки разработки самостоятельных программ. Улучшились общие навыки в программировании. Была сделана работающая программа, прошедшая полную отладку и готовая к эксплуатации.

# Список используемых источников

1. Основы алгоритмизации и ООП: методические указания к курсовой работе / Рязан. гос. радиотехн. ун-т; сост.: С.И. Елесина. – Рязань, 2020. – 20 с.
2. Односвязный линейный список на базе классов ООП [вебсайт] URL: <https://prog-cpp.ru/data-ols-oop/>
3. Как построить графики в Windows Forms. Компонент - Chart (Пример кода на C++/C#) [видеозапись] // YouTube. URL: <https://www.youtube.com/watch?v=y8sEeIQUfAI&t=1755s>
4. C++ перенос программы с консольного приложения на WinForms. (5. Пример с переносом класса) [видеозапись] // YouTube. URL: <https://www.youtube.com/watch?v=AN3eDimV5vU&t=816s>

# Приложение А

Код файла Class.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <string>

using namespace std;

static char ch;

static const int nameL = 45, nameK = 7, nameD = 10;

struct ABONENT {

int number;

char F[nameL];

char I[nameL];

char O[nameL];

char kafedr[nameK];

int group;

char perereg[2];

char pol[5];

char dolznost[nameD];

char knigi[2];

int data[3];

int god;

};

class Node

{

int field;

class Node\* ptr;

friend class List;

ABONENT inf;

};

class List

{

Node\* head; // Корень списка

int count = 0; // Количество узлов списка

Node\* Prev(Node\*); // Переход к предыдущему узлу

friend class Node;

public:

List() { head = NULL; } // Инициализация списка

int getCount() { return count; } // Получение количества узлов списка

bool isEmpty() { return head == NULL; } // Проверка, пуст ли список

ABONENT getValue(Node\* p) { return p->inf; } // Получение значения узла списка

void setValue(Node\* p, int val) { p->field = val; } //Установка значения узла списка

Node\* getFirst() { return head; } // Получение корневого узла списка

Node\* getLast(); // Получение последнего узла списка

void Clear(); // Очистка списка

Node\* Next(Node\*); // Переход к следующему узлу

Node\* Add(ABONENT abonent, Node\*);// Добавление узла списка

Node\* Delete(Node\*); // Удаление узла списка

ABONENT Print(int a); //Вывод значений узлов списка

void Swap(Node\*, Node\*); // Взаимообмен двух узлов

void SortByName(); //Сортировка по имени

bool SortByPerereg(int j); //Сортировка по непрошедшим перерегистрацию

bool SortByDolg(string kafedra, int j); //Сортировка по должникам кафедры

float Sotrud(); //кол-во Сотрудников

float Stud(); //кол-во Студентов

int SeekByNum(int number); //Поиск по номеру билета

void WriteFile(); //Сохранение в файл

void Readfile(char\* z); // Чтение из файла

};

Код файла Class.cpp

#include "pch.h"

#include "Class.h"

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

using namespace std;

using namespace kursVis;

List list;

Node\* List::Add(ABONENT abonent, Node\* node = NULL)

{

Node\* elem = new Node();

elem->inf = abonent;

count++;

if (node == NULL) // Добавление нового корня

{

if (head == NULL) {

elem->ptr = NULL;

head = elem;

}

else {

elem->ptr = head;

head = elem;

}

return elem;

}

elem->ptr = node->ptr; // Добавление узла после текущего

node->ptr = elem;

return elem;

}

Node\* List::Delete(Node\* node)

{

if (node == NULL) { return NULL; } // В списке нет узлов

count--;

if (node == head) // Удаление корневого узла

{

head = node->ptr;

delete node;

return head;

}

Node\* prev = Prev(node); // Удаление промежуточного узла

prev->ptr = node->ptr;

delete node;

return prev;

}

ABONENT List::Print(int a)

{

Node\* p = head;

int k = 0;

while (k != a) { p = Next(p); k++; }

return p->inf;

}

void List::Clear()

{

class Node\* p = head;

if (p == NULL) return;

do {

Node\* d = p;

p = Next(p);

delete d;

} while (p != NULL);

count = 0;

head = NULL;

}

Node\* List::getLast()

{

Node\* p = head;

while (Next(p) != NULL)

p = Next(p);

return p;

}

Node\* List::Next(Node\* node)

{

if (isEmpty()) return NULL;

return node->ptr;

}

Node\* List::Prev(Node\* node)

{

if (isEmpty()) return NULL;

if (node == head) return NULL;

Node\* p = head;

while (p->ptr != node)

p = p->ptr;

return p;

}

void List::Swap(Node\* node1, Node\* node2)

{

if (node1 == NULL || node2 == NULL) return;

if (node1 == node2) return;

if (node2->ptr == node1)

{

Node\* p = node1;

node1 = node2;

node2 = p;

}

Node\* prev1 = Prev(node1);

Node\* prev2 = Prev(node2);

Node\* next1 = Next(node1);

Node\* next2 = Next(node2);

if (next1 == node2)

{

if (prev1 != NULL)

prev1->ptr = node2;

else

head = node2;

node2->ptr = node1;

node1->ptr = next2;

return;

}

if (prev1 != NULL)

prev1->ptr = node2;

else

head = node2;

if (prev2 != NULL)

prev2->ptr = node1;

else

head = node1;

node2->ptr = next1;

node1->ptr = next2;

}

void List::SortByName()

{

for (int l = 0; l < count; l++) {

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

Node\* p1 = getLast(), \* p2 = Prev(p1);

for (int i = 0; i < count-1; i++) {

if (strcmp(p1->inf.F, p2->inf.F) < 0) {

Swap(p2, p1);

}

else {

if (strcmp(p1->inf.F, p2->inf.F) == 0) {

if (strcmp(p1->inf.I, p2->inf.I) < 0) {

Swap(p2, p1);

}

else {

if (strcmp(p1->inf.I, p2->inf.I) == 0) {

if (strcmp(p1->inf.O, p2->inf.O) < 0) {

Swap(p2, p1);

}

}

}

}

}

p1 = Prev(p1);

p2 = Prev(p2);

}

}

}

}

bool List::SortByPerereg(int j)

{

Node\* p = head;

int i = 0;

while (i != j) {

p = Next(p);

i++;

}

if (strcmp(p->inf.perereg, "N") == 0)

{

return true;

}

}

bool List::SortByDolg(string kafedra, int j)

{

Node\* p = head;

char kaf[nameK];

char\* cstr = new char[kafedra.size()];

strcpy(cstr, kafedra.c\_str());

int i = 0;

while (i != j) {

p = Next(p);

i++;

}

if ((strcmp(p->inf.kafedr, cstr) == 0) && (strcmp(p->inf.knigi, "Y") == 0))

{

return true;

}

}

float List::Sotrud() {

char dolznost[nameD] = "сотрудник";

Node\* p = head;

float sotrKol = 0;

do {

if (strcmp(p->inf.dolznost, dolznost) == 0) { sotrKol++; }

p = Next(p);

} while (p != NULL);

return sotrKol;

}

float List::Stud()

{

char dolznost[nameD] = "студент";

Node\* p = head;

int i = 0;

float studKol = 0;

do {

if (strcmp(p->inf.dolznost, dolznost) == 0) { studKol++; }

i++;

p = Next(p);

} while (p != NULL);

return studKol;

}

int List::SeekByNum(int number) {

Node\* p = head;

int i = 0;

do {

if (p->inf.number == number) { return i; }

i++;

p = Next(p);

} while (p != NULL);

}

void List::Readfile(char\* z)

{

try {

Node\* node = new Node();

FILE\* f = fopen(z, "r");

list.Clear();

int i = 0;

while (!feof(f))

{

ABONENT abonent;

ch = fscanf(f, "%u", &abonent.number);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.F);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.I);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.O);

std::fgetc(f);

ch = fscanf(f, "%u", &abonent.god);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.pol);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.kafedr);

std::fgetc(f);

ch = fscanf(f, "%u", &abonent.group);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.dolznost);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.perereg);

std::fgetc(f);

ch = fscanf(f, "%s", abonent.knigi);

std::fgetc(f);

ch = fscanf(f, "%u", &abonent.data[0]);

std::fgetc(f);

ch = fscanf(f, "%u", &abonent.data[1]);

std::fgetc(f);

ch = fscanf(f, "%u", &abonent.data[2]);

while (std::fgetc(f) != '\n' && !feof(f));

if (ch != EOF) list.Add(abonent);

i++;

}

std::fclose(f);

}

catch (...) {

MessageBox::Show("Файл не был открыт", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

void List::WriteFile()

{

Node\* p = head;

FILE\* f = fopen("buff.txt", "w");

for (int i = 1; i <= count; i++)

{

ABONENT abonent = p->inf;

char\* wF = abonent.F;

char\* wI = abonent.I;

char\* wO = abonent.O;

char\* wpol = abonent.pol;

char\* wknigi = abonent.knigi;

char\* wdolznost = abonent.dolznost;

char\* wkafedr = abonent.kafedr;

char\* wperereg = abonent.perereg;

unsigned int wnum = abonent.number;

unsigned int wgroup = abonent.group;

unsigned int wgod = abonent.god;

unsigned int wadata0 = abonent.data[0];

unsigned int wadata1 = abonent.data[1];

unsigned int wadata2 = abonent.data[2];

fprintf(f, "%u %s %u %s %s %u %s %s %s %u %u %u\n", wnum, wF, wI, wO, wgod, wpol, wkafedr, wgroup, wdolznost, wperereg, wknigi, wadata0, wadata1, wadata2);

p = Next(p);

}

fclose(f);

}

Код файла kursVis.cpp

#include "pch.h"

#include "MyForm.h"

#include "MyForm1.h"

#include <iostream>

#include <fstream>

#include <Windows.h>

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

int main(array<String ^> ^args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

kursVis::MyForm1 form;

Application::Run(% form);

}

Код файла MyForm1.h

#pragma endregion

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

MyForm^ frm = gcnew MyForm();

frm->Show();

this->Hide();

}

Код файла MyForm.cpp

#include "pch.h"

#include "MyForm.h"

#include <iostream>

#include <fstream>

#include <Windows.h>

#include <msclr\marshal\_cppstd.h>

#include <msclr/marshal.h>

#include <string>

#include "Class.h"

#include <cstring>

#include "cmath"

extern List list;

System::Void kursVis::MyForm::dataGridView1\_CellContentClick(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e)

{

return System::Void();

}

System::Void kursVis::MyForm::button\_zagruz\_Click(System::Object^ sender, System::EventArgs^ e) //Загрузить таблицу из файла

{

dataGridView1->Rows->Clear();

String^ Filename = ""; //Создаём переменную для хранения пути к файлу

if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {

Filename = openFileDialog1->FileName;

}

try {

if (Filename == "") {

MessageBox::Show(this, "Файл не был открыт", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

string newfilname = msclr::interop::marshal\_as<std::string>(Filename);

char\* z = const\_cast<char\*>(newfilname.c\_str());

list.Readfile(z);

TableFill();

diagBuild();

MessageBox::Show("Таблица загружена");

}

catch (Exception^ e) {

MessageBox::Show(this, "Файл не был открыт", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

System::Void kursVis::MyForm::button\_add\_Click(System::Object^ sender, System::EventArgs^ e) // Добавить строку

{

dataGridView1->Rows->Add();

}

System::Void kursVis::MyForm::button\_save\_Click(System::Object^ sender, System::EventArgs^ e) // Сохранить таблицу в файл

{

if (!dataGridView1->RowCount)

{

MessageBox::Show("В таблице нет строк!");

return;

}

try {

String^ Filename = ""; //Создаём переменную для хранения пути к файлу

if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {

Filename = openFileDialog1->FileName;

}

edit(Filename);

MessageBox::Show("Таблица сохранена");

}

catch (...) {

MessageBox::Show(this, "Файл не был сохранён", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

System::Void kursVis::MyForm::textBox1\_TextChanged(System::Object^ sender, System::EventArgs^ e)

{

}

System::Void kursVis::MyForm::button\_Search\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (!dataGridView1->RowCount)

{

MessageBox::Show("В таблице нет строк!");

return;

}

bool found = false;

String^ s1 = textBox1->Text;

try {

int num = Convert::ToInt32(s1);

int n = 0;

while (n != dataGridView1->RowCount) {

dataGridView1->Rows[n]->Visible = false;

n++;

}

try {

dataGridView1->Rows[list.SeekByNum(num)]->Visible = true;

found = true;

}

catch (...) {

MessageBox::Show("Абонента с таким номером не существует");

for (int i = 0; i < dataGridView1->RowCount; i++) {

dataGridView1->Rows[i]->Visible = true;

}

}

}

catch (...) {

MessageBox::Show("Вы ввели не число");

return;

}

}

System::Void kursVis::MyForm::таблицаToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

for (int i = 0; i < dataGridView1->RowCount; i++) {

dataGridView1->Rows[i]->Visible = true;

}

}

System::Void kursVis::MyForm::button\_DolgFak\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (comboBox\_Kafedra->Text == "") {

MessageBox::Show("Сначала выберите кафедру");

return;

}

if (!dataGridView1->RowCount)

{

MessageBox::Show("В таблице нет строк!");

return;

}

bool found = false;

String^ s1 = comboBox\_Kafedra->Text;

std::string s2 = msclr::interop::marshal\_as<std::string>(s1);

for (int i = 0; i < dataGridView1->RowCount; i++) {

if(list.SortByDolg(s2, i) == true){

dataGridView1->Rows[i]->Visible = true;

found = true;

}

else { dataGridView1->Rows[i]->Visible = false; }

}

if (found == false) {

MessageBox::Show("Нет должников на этой кафедре");

for (int i = 0; i < dataGridView1->RowCount; i++) {

dataGridView1->Rows[i]->Visible = true;

}

}

}

System::Void kursVis::MyForm::comboBox1\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

return System::Void();

}

System::Void kursVis::MyForm::button\_Perereg\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (!dataGridView1->RowCount)

{

MessageBox::Show("В таблице нет строк!");

return;

}

bool found = false;

for (int j = 0; j < dataGridView1->RowCount; j++) {

bool t = list.SortByPerereg(j);

if (list.SortByPerereg(j) == true) {

dataGridView1->Rows[j]->Visible = true;

found = true;

}

else { dataGridView1->Rows[j]->Visible = false; }

}

if (found == false) {

MessageBox::Show("Нет непрошедших перерегистрацию");

for (int i = 0; i < dataGridView1->RowCount; i++) {

dataGridView1->Rows[i]->Visible = true;

}

}

}

System::Void kursVis::MyForm::chart1\_Click(System::Object^ sender, System::EventArgs^ e)

{

}

System::Void kursVis::MyForm::CellEndEdit(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e)

{

String^ Filename = "buffer.txt"; //Создаём переменную для хранения пути к файлу

string newfilname = msclr::interop::marshal\_as<string>(Filename);

char\* z = const\_cast<char\*>(newfilname.c\_str());

edit(Filename);

list.Readfile(z);

diagBuild();

}

System::Void kursVis::MyForm::сортировкаПоФИОToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (!dataGridView1->RowCount)

{

MessageBox::Show("В таблице нет строк!");

return;

}

dataGridView1->Rows->Clear();

list.SortByName();

TableFill();

}

System::Void kursVis::MyForm::diagBuild()

{

if (!dataGridView1->RowCount)

{

return;

}

this->chart1->Series[0]->Points->Clear();

this->chart1->Series[0]->Points->Add(list.Stud());

this->chart1->Series[0]->Points[0]->AxisLabel = Convert::ToString(round((list.Stud() / dataGridView1->RowCount \* 100) \* 100) / 100) + "%";

this->chart1->Series[0]->Points[0]->LegendText = "Студенты";

this->chart1->Series[0]->Points->Add(list.Sotrud());

this->chart1->Series[0]->Points[1]->AxisLabel = Convert::ToString(round((list.Sotrud() / dataGridView1->RowCount \* 100) \* 100) / 100) + "%";

this->chart1->Series[0]->Points[1]->LegendText = "Сотрудники";

}

System::Void kursVis::MyForm::edit(String ^ Filename)

{

StreamWriter^ myWriter = gcnew StreamWriter(Filename, false, System::Text::Encoding::GetEncoding(1251));

for (int i = 0; i < dataGridView1->RowCount; i++) {

for (int j = 0; j < dataGridView1->ColumnCount; j++) {

myWriter->Write(Convert::ToString(dataGridView1->Rows[i]->Cells[j]->Value));

myWriter->Write(" ");

}

myWriter->WriteLine();

}

myWriter->Close();

}

System::Void kursVis::MyForm::TableFill() {

int a = 0;

while (a != list.getCount()) {

ABONENT abonent = list.Print(a);

String^ strF = gcnew String(abonent.F);

String^ strI = gcnew String(abonent.I);

String^ strO = gcnew String(abonent.O);

String^ strPol = gcnew String(abonent.pol);

String^ strKafedr = gcnew String(abonent.kafedr);

String^ strDolznost = gcnew String(abonent.dolznost);

String^ strPerereg = gcnew String(abonent.perereg);

String^ strKnigi = gcnew String(abonent.knigi);

dataGridView1->Rows->Add(abonent.number, strF + " " + strI + " " + strO, abonent.god, strPol, strKafedr, abonent.group, strDolznost, strPerereg, strKnigi, abonent.data[0] + "." + abonent.data[1] + "." + abonent.data[2]);

a++;

}

}

System::Void kursVis::MyForm::button\_delete\_Click(System::Object^ sender, System::EventArgs^ e) // Удалить строку

{

//удалить строку

if (!dataGridView1->RowCount)

{

MessageBox::Show("В таблице нет строк!");

return;

}

if (dataGridView1->SelectedRows->Count > 0) {

for (int i = 0; i < dataGridView1->SelectedRows->Count; i++) {

int index = dataGridView1->SelectedRows[i]->Index;

try {

dataGridView1->Rows->RemoveAt(index);

}

catch (...) {

MessageBox::Show("Не могу удалить строку с индексом " + index);

}

}

}

else {

int index;

try {

index = dataGridView1->CurrentCell->RowIndex;

}

catch (...) {}

if (index != -1) {

try { dataGridView1->Rows->RemoveAt(index); }

catch (...) {

MessageBox::Show("Не могу удалить строку с индексом " + index);

}

}

else MessageBox::Show("Выберите строкy для удаления");

}

String^ Filename = "buffer.txt";

string newfilname = msclr::interop::marshal\_as<std::string>(Filename);

char\* z = const\_cast<char\*>(newfilname.c\_str());

edit(Filename);

list.Readfile(z);

diagBuild();

}

System::Void kursVis::MyForm::выходToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) // Выход из приложения

{

Application::Exit();

}

System::Void kursVis::MyForm::справкаToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) // Справка

{

MessageBox::Show("Данная программа позволяет работать со списком абонентов библиотеки. Она сделана с использованием языка программирования С++");

return System::Void();

}

System::Void kursVis::MyForm::openFileDialog1\_FileOk(System::Object^ sender, System::ComponentModel::CancelEventArgs^ e)

{

}