САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Разработка электронной картотеки

Студент гр. 9305 _____ Бессонов Н.Е

Преподаватель _____ Перязева Ю.В

Санкт-Петербург
2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Бессонов Н.Е

Группа 9305

Тема работы : Разработка электронной картотеки

Исходные данные:

Подготавливается файл, содержащий определенную информацию, который затем считывается программой.

Содержание пояснительной записки:
Перечисляются требуемые разделы пояснительной записки (обязательны разделы «Содержание», «Введение», «Заключение», «Список использованных источников»)

Предполагаемый объем пояснительной записки:
Не менее 00 страниц.

Дата выдачи задания: 01.05.2020

Дата сдачи реферата: 05.06.2020

Дата защиты реферата: 06.06.2020

Студент _____ Бессонов Н.Е

Преподаватель _____ Перязева Ю.В

АННОТАЦИЯ

Электронные картотеки сейчас широко используются, в них могут находятся сотни или тысячи человек, и работа с таким большим объемом информации, поиск в ней, редактирование, без помощи программ может занимать очень много времени, поэтому целью данной курсовой работы было создать программу, которая создает картотеку, и помогает пользователю работать в ней. В программе доступно множество функций, позволяющих облегчить взаимодействие пользователя с картотекой, выдавая практически мгновенно нужный результат.

SUMMARY

Electronic file cabinets are now widely used, they can contain hundreds or thousands of people, and working with such a large amount of information, searching in it, editing, without the help of programs can take a very long time, so the goal of this course work was to create a program that creates a file cabinet , and helps the user to work in it. The program has many functions available that make it easier for the user to interact with the file cabinet, giving almost instantly the desired result.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является написать программу, позволяющую облегчить работу пользователя с электронной картотекой, за счет встроенных функций по редактированию, добавлению, поиску и другим действиям с картотекой.

Задачи:

- Разработать электронную картотеку
- Изучить линейные списки, для использования их в картотеке
- Продумать интерфейс общения пользователя с программой
- Продумать и создать модули по взаимодействию пользователя с картотекой
- Написать программу

1. ЭЛЕКТРОННАЯ КАРТОТЕКА

Электронная картотека – это один из способов взаимодействия с базами данных, в которых хранятся упорядоченные данные с общим содержанием.

Данная программа позволит пользователю ускорить процесс общения с электронной картотекой. Теперь доступен быстрый поиск элементов по заданным параметрам, сортировка, копирование и другие действия с элементами.

Программа написана таким образом, что даже если пользователь введет не верное значение, или случайно нажмет не туда, то он сможет проделать все операции заново. При каждом действии пользователь видет картотеку, что позволяет более удобно взаимодействовать с ней. Если по какой-либо причине какие-либо функции не доступны (например, список пуст), то программа сообщит об этом пользователю, и предложит продолжить работу дальше.

2. ПРОГРАМНАЯ РЕАЛИЗАЦИЯ

2.1. Описание работы программы

После запуска программы на экран пользователю выводится список операций, которыми он может пользоваться при работе с картотекой, однако сама картотека еще не создана, как это сделать выбирает сам пользователь. Он может считать ее из файла формата txt или csv, либо может заполнять ее вручную.

После того как картотека заполнена, пользователь может совершать с ней различные манипуляции, за каждую из которых отвечает своя функция

Список доступных пользователю функций:

- Считать картотеку из файла
- Добавить элемент в конец картотеки
- Добавить элемент в начало картотеки
- Удалить первый/последний элемент
- Поиск элемента по значению
- Сортировка картотеки по различным параметрам
- Редактирование ячеек картотеки
- Копирование определенных объектов в картотеки и запись их в файл
- Запись картотеки в файл
- Реверс картотеки

После выбора действия запускается та или иная функция. Все основные функции разнесены по файлам, что облегчает взаимодействие с ними, а также добавляет программе структурированности.

2.2. Описание структур данных

Структура узла NODE

Имя поля	Тип	Назначение
Name	char*	Имя сдающего экзамен
Data	char*	Дата экзамена
Age	int	Возраст сдающего
Time	char*	Время сдачи экзамена
Maxpoint	int	Максимальное количество баллов
Error	float	Количество баллов снятых за ошибки
Resultpoint	float	Итоговое количество баллов
Result	char*	Результат сдал/не сдал
Next	struct list*	Следующий элемент списка

Структура головы HEAD

Имя поля	Тип	Назначение
count	int	Количество элементов в списке
First	struct list*	Указатель на первый узел
last	struct list*	Указатель на последний узел

2.3. Описание функций

`main`

Описание:

Прототип: `int main()`

Пример вызова: `main()`

Возвращаемое значение: отсутствует

`puts_line`

Описание: получение новой информации с клавиатуры и преобразование ее в структуру

Прототип: `NODE* puts_line()`

Пример вызова: `puts_line()`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Локальная	ptr	NODE*	Новая структура
Локальная	S	char	Получаемая от пользователя строка
Локальная	K	int	Количество символов в строке

Возвращаемое значение: заполненная структура типа NODE*

`split`

Описание: разбиение строки по символу разделителю

Прототип: `NODE* split(char* s1, int n1)`

Пример вызова: `split(s, k)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	s1	char*	Входящая строка
Формальный аргумент	n1	int	Длина строки
Локальная	`s2`	`char**`	Массив строк, заполненный слова, после разбиения входящей строки по символу разделителю
Локальная	sep	char	Символ разделитель
Локальная	i, f, j, p, k, m	int	Служебные переменные
Локальная	len	int	Количество символов во входящей строке
Локальная	`str`	`NODE*`	Новая структура, заполненная словами из входящей строки

Возвращаемое значение: заполненная структура типа NODE*

`struct_fill`

Описание: создание и заполнение новой структуры словами из входящего массива строк

Прототип: `NODE* struct_fill(char** str)`

Пример вызова: `struct_fill(s2);`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Локальная	`str`	`NODE*`	Новая структура, заполненная словами из входящей строки
Формальный аргумент	Str	Char**	Массив строк

Возвращаемое значение: заполненная структура типа NODE*

`menu`

Описание: Отвечает за своевременную остановку программы

Прототип: `void menu()`

Пример вызова: `menu()`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Локальная	`STOP`	str	Идентификатор остановки работы программы

Возвращаемое значение: отсутствует

`input`

Описание: Функция создана для работы с пользователем, она является основным меню, пользователь вводит то что он хочет сделать со списком, а данная функция запускает отвечающее за это функцию

Прототип: `int input()`

Пример вызова: `input()`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Локальная	k	str	Идентификатор остановки работы функции
Локальная	S1	char	Строка для получения команды от пользователя
Локальная	head	HEAD*	Голова создаваемого списка
Локальная	ptr	NODE*	Один элемент списка
Локальная	Z, key	int	Вспомогательные переменные

Возвращаемое значение: отсутствует

`output`

Описание: Обертка для печати списка

Прототип: `void output(HEAD* head)`

Пример вызова: `output(head)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка

Возвращаемое значение: отсутствует

`print_menu`

Описание: Функция, показывающая пользователю, в каком порядке нужно вводить информацию

Прототип: `void print_menu()`

Пример вызова: `print_menu()`

Возвращаемое значение: отсутствует

`printNODE`

Описание: Отвечает за печать всего списка

Прототип: `void printNODE(HEAD* ptr)`

Пример вызова: `printNODE(head)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	ptr1	NODE*	Переменная для печати

Возвращаемое значение: отсутствует

`printstructr`

Описание: Отвечает за печать одного элемента списка

Прототип: `void printstructr(NODE* id1)`

Пример вызова: `printstructr(ptr1)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	ptr1	NODE*	Переменная для печати

Возвращаемое значение: отсутствует

`add_last_node`

Описание: Добавляет элемент на последнее место в списке

Прототип: `void add_last_node(HEAD* head, NODE* ptr)`

Пример вызова: `add_last_node(head, puts_line())`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	str	NODE*	Указатель на новый элемент, который необходимо добавить в список

Возвращаемое значение: отсутствует

`add_first_node`

Описание: Добавляет элемент на первое место в списке

Прототип: `void add_first_node(HEAD* head, NODE* ptr)`

Пример вызова: `add_first_node(head, puts_line())`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	str	NODE*	Указатель на новый элемент, который необходимо добавить в список
Локальная	q	NODE*	Указатель на первый элемент списка
Локальная	I	int	Счетчик

Возвращаемое значение: отсутствует

`delet_last_node`

Описание: Удаление последнего элемента в списке

Прототип: `void delet_last_node(HEAD* head)`

Пример вызова: `delet_last_node(head);`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	last	NODE*	Указатель на последний элемент списка
Локальная	q	NODE*	Указатель на первый элемент списка

Возвращаемое значение: отсутствует

`delet_first_node`

Описание: Удаление первого элемента в списке

Прототип: `void delet_first_node(HEAD* head)`

Пример вызова: `delet_ first_node(head);`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	i	int	Счетчик

Возвращаемое значение: отсутствует

`add_from_file`

Описание: Считывание данных полей из файла

Прототип: `void add_from_file(HEAD* head)`

Пример вызова: `add_from_file (head);`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальный	N	Int	Количество строк в файле
Локальный	K	Int	Количество символов в строке
Локальный	Ptr	NODE*	Заполняемая структура
Локальный	S	Char	Получаемая из файла строка
Локальный	finame	char	Имя файла

Возвращаемое значение: отсутствует

`revers`

Описание: меняет направление списка

Прототип: `void revers(HEAD* head, int n)`

Пример вызова: `revers(head, n)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	n	int	Вспомогательное значение
Локальная	q	NODE*	Указатель на первый элемент списка
Локальная	I	int	Счетчик

Возвращаемое значение: отсутствует

Sort_menu

Описание: меню выбора параметров сортировки

Прототип: `void sort_menu(HEAD* head)`

Пример вызова: `sort_menu(head)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	Obj	int	Выбор параметра сортировки
Локальная	Way	int	Выбор возрастающая или убывающая сортировка
Локальная	I	int	Счетчик
Локальная	S1	char	Вводимая переменная
Локальная	q	NODE	Вспомогательная переменная

Возвращаемое значение: отсутствует

`Sort`

Описание: сортировка списка по заданным параметрам

Прототип: `void sort(HEAD* head, int n, int (*funcName)(NODE* p, int way))`

Пример вызова: `sort(head, way, by_age);`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	flag	int	Точка остановки
Локальная	k	int	Параметр
Локальная	I	int	Счетчик
Локальная	S1	char	Вводимая переменная
Локальная	P1	NODE	Первый элемент
Локальная	Q	NODE	Вспомогательный параметр
Локальная	B	NODE	Текущий элемент
Локальная	C	NODE	Следующий элемент
Локальная	d	NODE	Через один элемент
Локальная	q	NODE	Вспомогательная переменная

Возвращаемое значение: отсутствует

`By_id`

Описание: вызывается при сортировке по id, возвращает 1 или -1 в зависимости от выбранных настроек сортировки

Прототип: `int by_id(NODE* p, int way)`

Пример вызова: `by_id( p,  way)`

Возвращаемое значение: 1 или -1

`By_name`

Описание: вызывается при сортировке по name, возвращает 1 или -1 в зависимости от выбранных настроек сортировки

Прототип: `int by_name(NODE* p, int way)`

Пример вызова: `by_name( p,  way)`

Возвращаемое значение: 1 или -1

`By_age`

Описание: вызывается при сортировке по age, возвращает 1 или -1 в зависимости от выбранных настроек сортировки

Прототип: `int by_age(NODE* p, int way)`

Пример вызова: `by_age( p,  way)`

Возвращаемое значение: 1 или -1

`By_exam_time`

Описание: вызывается при сортировке по age, возвращает 1 или -1 в зависимости от выбранных настроек сортировки

Прототип: `int by_exam_time(NODE* p, int way)`

Пример вызова: `by_exam_time( p,  way)`

Возвращаемое значение: 1 или -1

`By_RB`

Описание: вызывается при сортировке по result ball, возвращает 1 или -1 в зависимости от выбранных настроек сортировки

Прототип: `int by_RB (NODE* p, int way)`

Пример вызова: `by_RB( p,  way)`

Возвращаемое значение: 1 или -1

`Edit_menu`

Описание: меню выбора параметров редактирования той или иной карточки

Прототип: `void edit_menu(HEAD* head)`

Пример вызова: `edit_menu(head)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	num	int	Порядковый номер пользователя, которого необходимо редактировать
Локальная	area	int	Номер ячейки для редактирования
Локальная	f	float	Счетчик
Локальная	q	NODE	Вспомогательная переменная

Возвращаемое значение: отсутствует

`Edit_char`

Описание: изменение значения ячейки типа char

Прототип: `void edit_char(NODE* q, int n)`

Пример вызова: `edit_char(q, n)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	q	NODE	Структура для изменения
Формальный аргумент	т	int	Номер ячейки для редактирования
Локальная	S1	char	Изменяемое значение
Локальная	s	char	Вспомогательная переменная

Возвращаемое значение: отсутствует

`Copy_element`

Описание: меню выбора для копирования элементов, создания нового списка и записи их в файл

Прототип: `void copy_element(HEAD* head)`

Пример вызова: `copy_element(head)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	n	int	Проверка на правильность ввода
Локальная	count	int	Количество номеров для копирования
Локальная	i	int	Счетчик
Локальная	num	Int*	Номера копируемых элементов

Возвращаемое значение: отсутствует

`Bubble_sort`

Описание: сортировка элементов

Прототип: `void bubbleSort(int* num, int size)`

Пример вызова: `bubbleSort(num, size)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	num	int *	Массив для сортировки
Формальный аргумент	Size	int	Размер массива
Локальная	flag	int	Параметр остановки

Возвращаемое значение: отсутствует

Select_id

Описание: выбор и копирование элемента с заданным id

Прототип: `NODE* select_id(HEAD* head, int k)`

Пример вызова: `select_id( head, k)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	k	int	Номер id
Локальная	count	int	Количество номеров
Локальная	p	int	Id текущего элемента
Локальная	i	Int*	счетчик
Локальная	Ptr	NODE*	Вспомогательная структура
Локальная	Ptr1	NODE*	Вспомогательная стурктура

Возвращаемое значение: скопированная структура

File_print

Описание: Записывает список в файл

Прототип: `void file_print(HEAD* head)`

Пример вызова: `file_print(head)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Вспомогательная структура
Локальная	finame	char	Имя файла для записи
Локальная	I	Int	Счетчик
Локальная	count	Int	Количество переменных
Локальная	id1	NODE*	Вспомогательная стурктура

Возвращаемое значение: отсутствует

Search

Описание: поиск элементов по заданным параметрам

Прототип: `void search(HEAD* head)`

Пример вызова: `search(head)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Локальная	k	int	Параметр
Локальная	New_head	HEAD*	Указатель на новый список
Локальная	New_ptr	NODE*	Элемент нового списка
Локальная	s	int	Параметр записи в файл

Возвращаемое значение: отсутсвует

Search_settings_1

Описание: поиск элементов в колонке name

Прототип: `void search_settings_1(HEAD* head, HEAD* new_head, int n)`

Пример вызова: `search_settings_1(HEAD* head, HEAD* new_head, int n)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	New_head	HEAD*	Указатель на голову нового списка
Формальный аргумент	s	char	Параметр поиска
Локальная	Q	NODE*	Вспомогательная переменная
Локальная	New_ptr	NODE*	Элемент нового списка
Локальная	m	int	Ошибки

Возвращаемое значение: отсутсвует

Search_settings_2

Описание: поиск элементов в колонке date

Прототип: `void search_settings_2(HEAD* head, HEAD* new_head, int n)`

Пример вызова: `search_settings_2(HEAD* head, HEAD* new_head, int n)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	New_head	HEAD*	Указатель на голову нового списка
Формальный аргумент	s	char	Параметр поиска
Локальная	Q	NODE*	Вспомогательная переменная
Локальная	New_ptr	NODE*	Элемент нового списка
Локальная	m	int	Ошибки

Возвращаемое значение: отсутсвует

Search_settings_3

Описание: поиск элементов в колонке age

Прототип: `void search_settings_3(HEAD* head, HEAD* new_head, int n)`

Пример вызова: `search_settings_3(HEAD* head, HEAD* new_head, int n)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	New_head	HEAD*	Указатель на голову нового списка
Формальный аргумент	k	int	Параметр поиска
Локальная	Q	NODE*	Вспомогательная переменная
Локальная	New_ptr	NODE*	Элемент нового списка
Локальная	m	int	Ошибки

Возвращаемое значение: отсутсвует

Search_settings_4

Описание: поиск элементов в колонке error

Прототип: `void search_settings_4(HEAD* head, HEAD* new_head, int n)`

Пример вызова: `search_settings_4(HEAD* head, HEAD* new_head, int n)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	New_head	HEAD*	Указатель на голову нового списка
Формальный аргумент	k	float	Параметр поиска
Локальная	Q	NODE*	Вспомогательная переменная
Локальная	New_ptr	NODE*	Элемент нового списка
Локальная	m	int	Ошибки

Возвращаемое значение: отсутсвует

Search_settings_5

Описание: поиск элементов в колонке result point

Прототип: `void search_settings_5(HEAD* head, HEAD* new_head, int n)`

Пример вызова: `search_settings_5(HEAD* head, HEAD* new_head, int n)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	head	HEAD*	Указатель на голову списка
Формальный аргумент	New_head	HEAD*	Указатель на голову нового списка
Формальный аргумент	k	float	Параметр поиска
Локальная	Q	NODE*	Вспомогательная переменная
Локальная	New_ptr	NODE*	Элемент нового списка
Локальная	m	int	Ошибки

Возвращаемое значение: отсутсвует

Error

Описание: поиск элементов в колонке result point

Прототип: `void error()`

Пример вызова: `error()`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Локальная	m	int	Ошибки

Возвращаемое значение: отсутсвует

Struct_copy

Описание: создание дубликата структуры

Прототип: `NODE* struct_copy(NODE* ptr)`

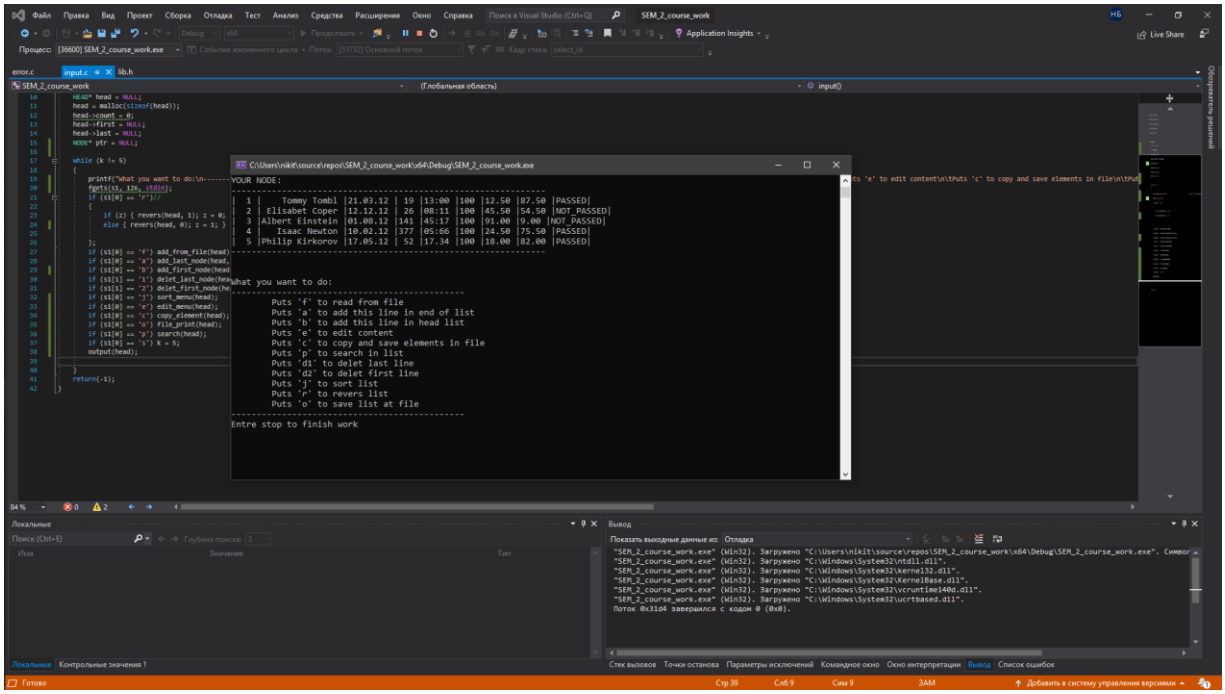
Пример вызова: `struct_copy(ptr)`

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальный аргумент	Ptr	NODE*	Вспомогательная структура
Локальная	Ptr1	NODE*	Вспомогательная стурктура

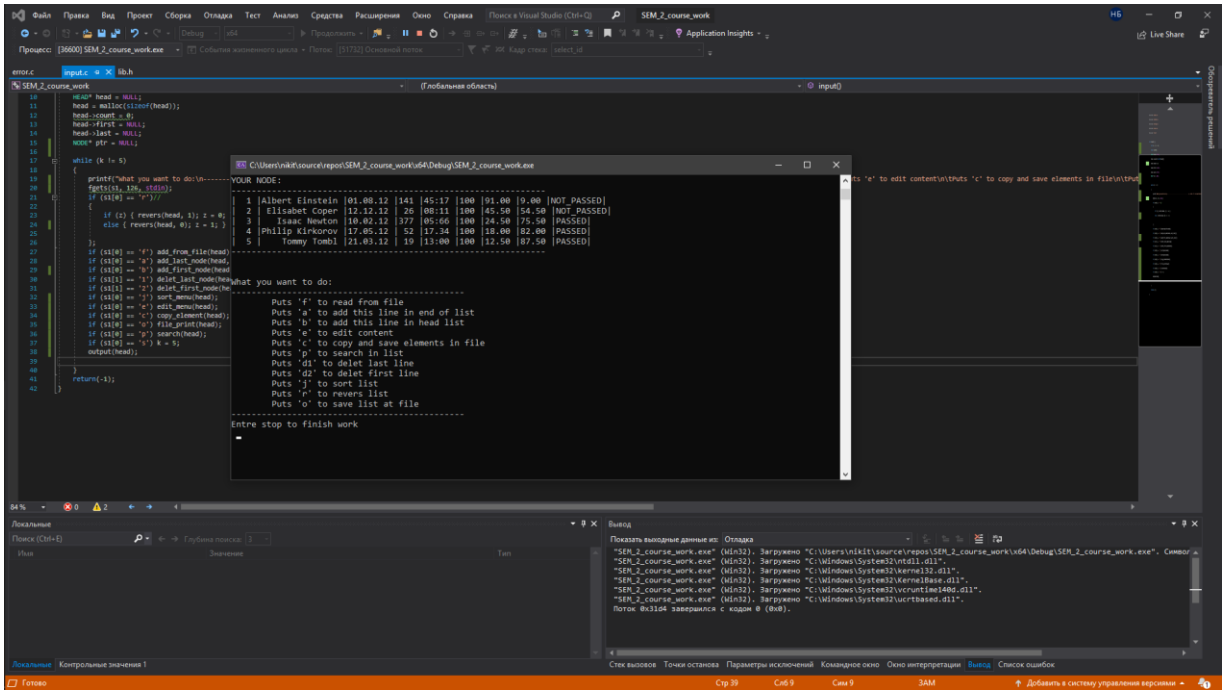
Возвращаемое значение: дубликат стурктуры

2.4. Пример работы программы

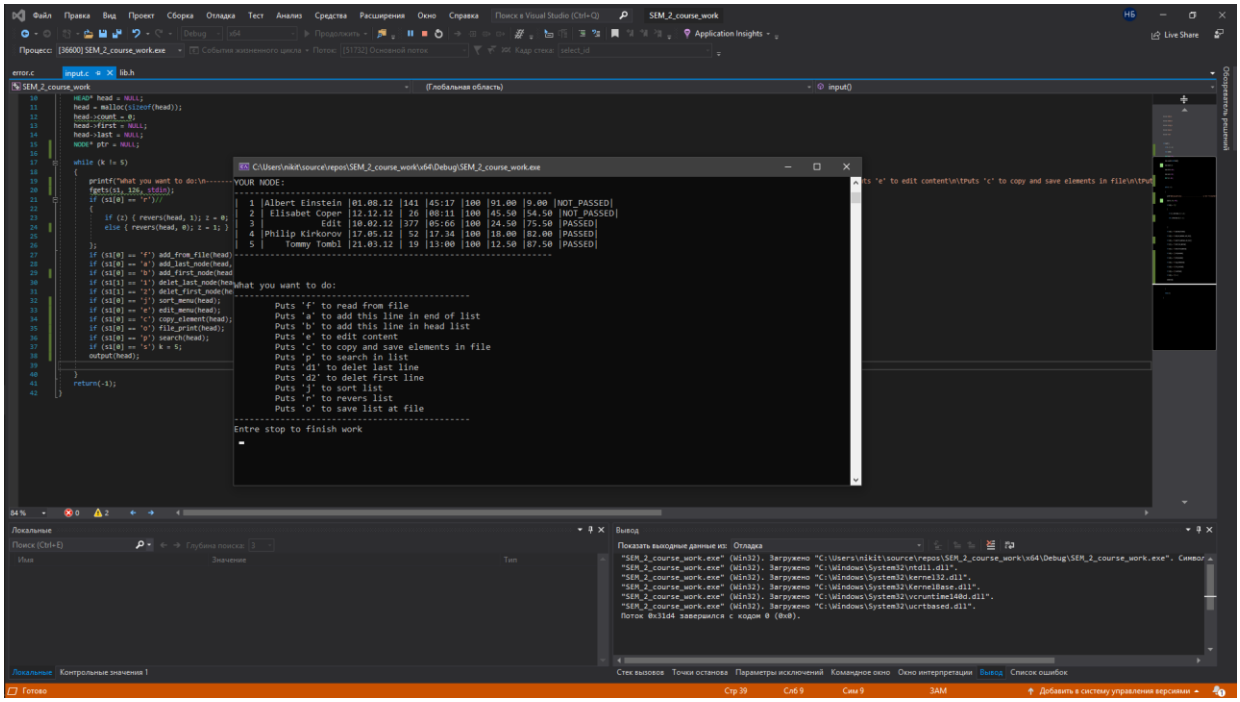
1. Считывание из файла text2.txt



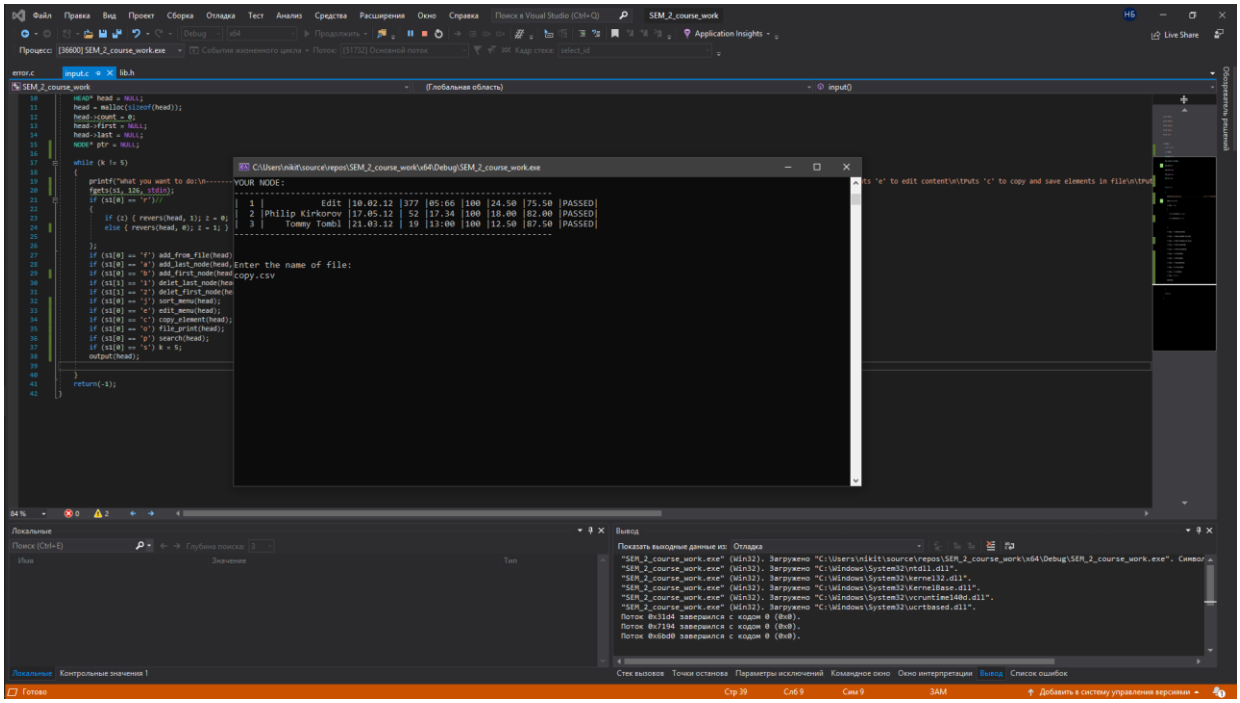
2. Сортировка картотеки по имени по убыванию



3. Изменение значения поля картотеки



4. Копирование элементов и запись их в файл



ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была создана электронная картотека, в основе которой лежит односвязный список. Были реализованы функции для работы и редактирования картотеки, такие как копирование элементов, редактирование ячеек, сортировка, поиск по значению и некоторые другие. В программе используется удобный интерфейс, способный помочь пользователю быстро и комфортно взаимодействовать с картотекой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Реалтзация односвязного Линейного списка в си
   https://tim4ous.com/realizatsiya-odnosvyaznogo-lineynogo-spiska-v-si/

2. Односвязный список
   https://learnc.info/adt/linked_list.html

3. Реализация структур данных си
   https://learnc.info/adt/

ПРИЛОЖЕНИЕ А
СХЕМА ВЫЗОВОВ ФУНКЦИЙ