МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное

учреждение высшего образования

«Казанский национальный исследовательский технический

университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

Институт компьютерных технологий и защиты информации

(наименование института (факультета), филиала)

\_\_Кафедра Прикладной Математики и Информатики \_\_

(наименование кафедры)

\_\_\_\_\_\_\_\_09.03.04 «Программная инженерия» \_\_\_\_

(шифр и наименование направления подготовки (специальности))

пояснительная записка

к Курсовой работе

по дисциплине:

«Структуры и алгоритмы обработки данных»

на тему: «Комбинированные структуры данных»

Вариант № 26.5

Обучающийся 4211 ГильфановаК.М.

(номер группы) (подпись, дата) (Ф.И.О.)

Руководитель доцент каф. ПМИ Сотников С.В.

(должность) (Ф.И.О.)

Курсовая работа зачтена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Казань 2021

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc38566255)

[1. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc38566256)

[1.1 Цель работы 4](#_Toc38566257)

[1.2 Исходные данные 4](#_Toc38566258)

[1.3 Ожидаемый результат 4](#_Toc38566259)

[1.4 Требования к реализации 5](#_Toc38566260)

[2. ОПИСАНИЕ РАЗРАБОТАННЫХ КЛАССОВ. АЛГОРИТМЫ РЕАЛИЗАЦИИ ОСНОВНЫХ ОПЕРАЦИЙ 6](#_Toc38566261)

[2.1 Реализация стека на статическом распределении памяти 6](#_Toc38566262)

[2.2 Реализация двунаправленного упорядоченного списка на динамическом распределении памяти 8](#_Toc38566263)

[2.3 Краткое описание разработанных классов 12](#_Toc38566264)

[2.4 Схема структуры проекта (взаимодействия классов) 13](#_Toc38566265)

[3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 14](#_Toc38566266)

[4. РУКОВОДСТВО ПРОГРАММИСТА 21](#_Toc38566267)

[ВЫВОД 25](#_Toc38566268)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 26](#_Toc38566269)

[ПРИЛОЖЕНИЕ. ЛИСТИНГИ ВСЕХ ФАЙЛОВ ПРОЕКТА 27](#_Toc38566270)

# ВВЕДЕНИЕ

В данной курсовой работе рассматривается реализация структуры данных “Статическая очередь неупорядоченных статических списков”.

Программа реализована на основе принципов объектно-ориентированного программирования (ООП) с использованием языка программирования С++ .

В пояснительной записке к курсовой работе описана постановка задачи, которую необходимо выполнить, алгоритм реализации программы, руководство пользователя и руководство программиста, включающие инструкции по работе с программой, объяснение программной реализации основных алгоритмов для возможности их совершенствования и последующего использования в других проектах.

В приложении к пояснительной записке включены листинги исходных кодов всех файлов проекта.

# 1. ПОСТАНОВКА ЗАДАЧИ

## 1.1 Цель работы

Программная реализация комбинированной структуры данных – Статическая очередь неупорядоченных статических списков – на основе объектно-ориентированного подхода, повторение и закрепление полученных знаний в ходе изучения курсов «Структуры и алгоритмы обработки данных» и «Объектно-ориентированное программирование», выработка навыков владения языком программирования.

## 1.2 Исходные данные

Вариант 26 задания на курсовую работу по ООП и САОД: Статическая очередь неупорядоченных статических списков .Вариант 5 информационного наполнения контейнера: *Школа(номер)-композиция классов(имя класса).Класс-композиция учеников(фамилия, год рождения).* Главная структура (в скобках – возможное свойство) – это набор подчиненных структур (в скобках – возможное свойство, связываемое с каждой структурой); подчиненная структура – это набор элементов нижнего уровня (в скобках – два возможных информационных свойства, для хранения и обработки которых должны использоваться соответствующие объекты).

Возможные языки программирования: Delphi Pascal, C, C++.

## 1.3 Ожидаемый результат

Программно-реализованная комбинированная структура данных (согласно варианту 26 задания на курсовую работу и варианту 5 наполнения контейнера) с использованием объектного подхода и языка программирования C++, согласно всем требованиям к реализации, описанным в следующем подпункте.

Приобретение и усвоение новых навыков программирования и работы со структурами данных. Закрепление умений применения объектно-ориентированного подхода в программировании. Улучшение навыков работы с языком программирования C++.

## 1.4 Требования к реализации

1. Реализация комбинированной структуры на основе объектного подхода;

2. Реализация всех необходимых операций (добавление и удаление в основной и присоединенной структурах, поиск в списке);

3. Возможность сохранения всей структуры по внешнем файле (текстовом или XML) с обратной загрузкой;

4. Реализация структуры для хранения и обработки данных конкретной информационной задачи;

5. Именование типов, структур и их полей, классов и их свойств, методов в соответствии с конкретной информационной задачей.

# 2. ОПИСАНИЕ РАЗРАБОТАННЫХ КЛАССОВ. АЛГОРИТМЫ РЕАЛИЗАЦИИ ОСНОВНЫХ ОПЕРАЦИЙ

## 2.1 Реализация очереди на статическом распределении памяти

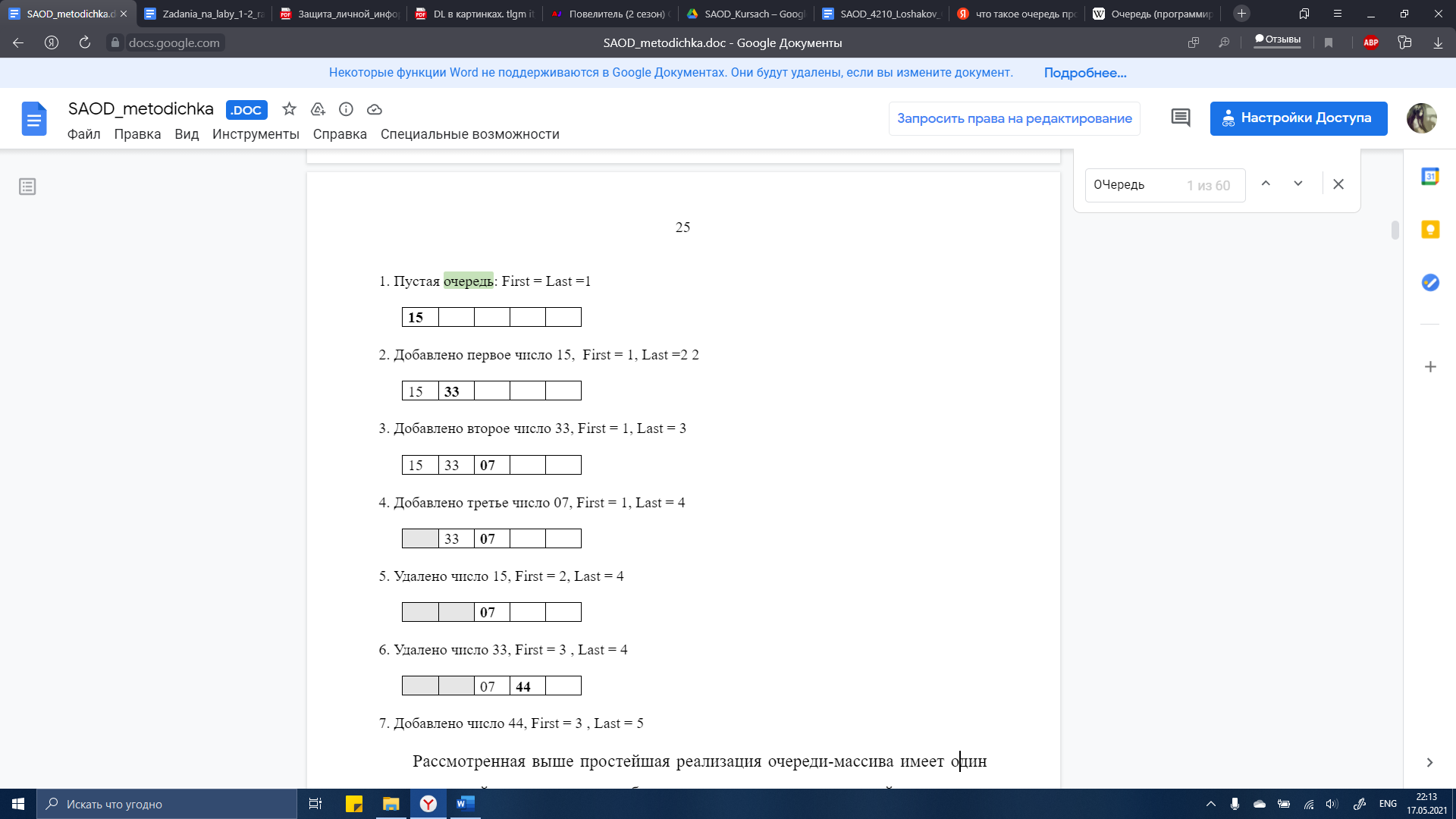
Очередь — [абстрактный тип данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) с дисциплиной доступа к элементам «первый пришёл — первый вышел» ([FIFO](https://ru.wikipedia.org/wiki/FIFO), [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *first in, first out*). Добавление элемента возможно лишь в конец очереди, выборка — только из начала очереди, при этом выбранный элемент из очереди удаляется.

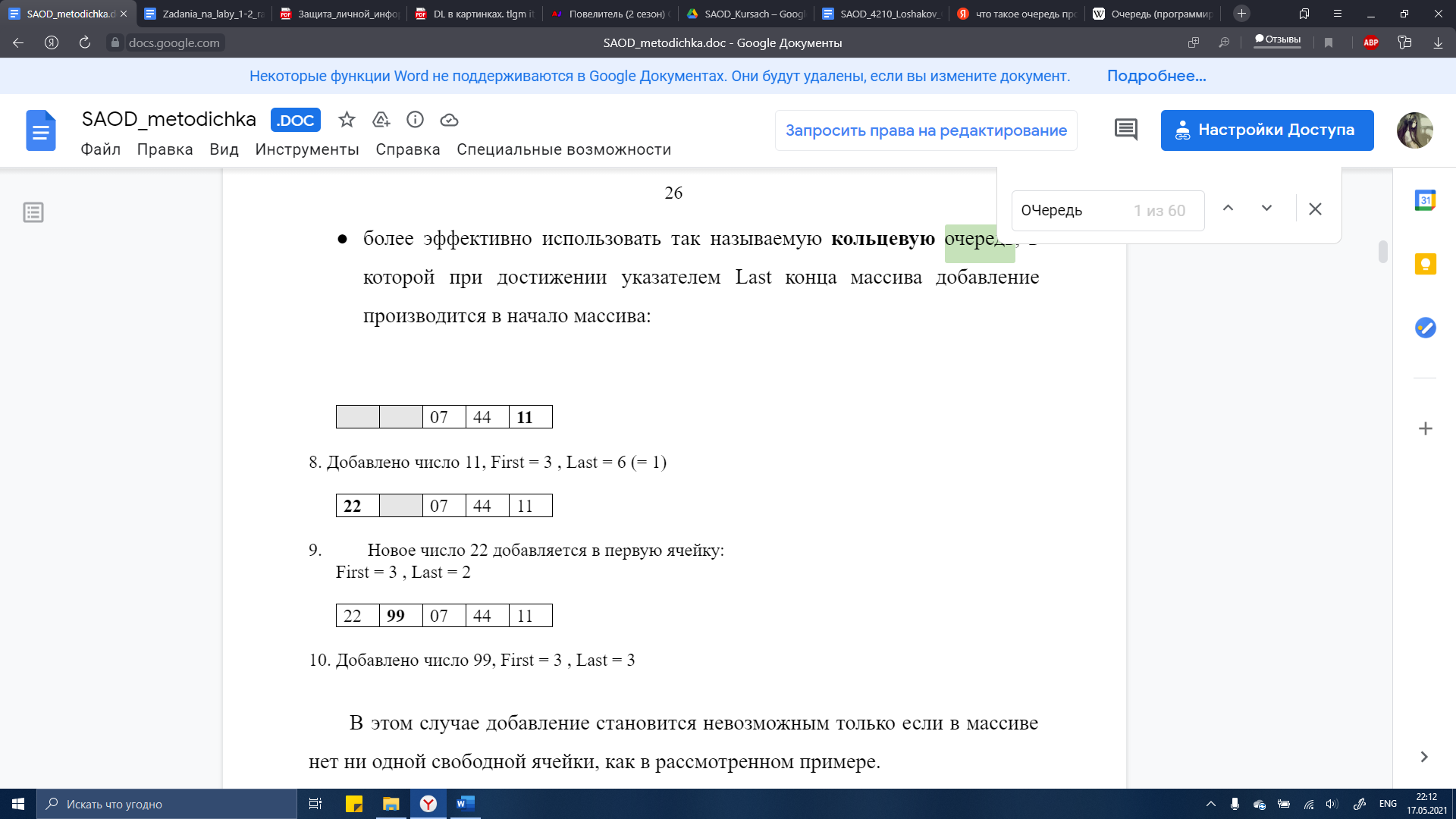
Программная реализация очереди возможна двумя способами:

* статически с помощью указателей Last и First
* динамически с помощью механизма указателей

В курсовой работе был использован первый способ. Для статической реализации структуры Очередь необходимо объявить указатель на первый и последний элемент .

Освобождающиеся при удалении ячейки в начале массива не используются при последующих добавлениях, и поэтому при интенсивном использовании очереди быстро может возникнуть ситуация, когда указатель Last выходит за пределы массива, тогда как в начале массива есть свободные ячейки. Для решения этой проблемы можно использовать кольцевую очередь, в случае если очередь переполнена новые элементы будут добавляться в начало.





С очередью связываются две основные операции: добавление (вталкивание, Add) элемента в стек и удаление (выталкивание, del) элемента из стека.

Само **добавление элемента** в очередь выполняется следующим образом:

* проверить возможность добавления (в массиве есть свободные ячейки?)
* добавить элемент в массив по индексу Last
* изменить указатель Last на 1
* если Last выходит за пределы массива, то установить Last в 1
* увеличить счетчик числа элементов в очереди

**Удаление элемента** из очереди:

* проверить возможность удаления (в очереди есть элементы?)
* извлечь элемент из массива по индексу First и выполнить с ним необходимые действия
* увеличить указатель First на 1
* если First выходит за пределы массива, то установить First в 1
* уменьшить счетчик числа элементов в очереди

## 2.2 Реализация неупорядоченного списка на статической распределении памяти

Линейный список – это набор связанных однотипных элементов, в котором каждый элемент каким-то образом определяет следующий за ним элемент. Добавление нового элемента возможно в любом месте списка, также можно удалить любой элемент списка.

В отличие от стека и очереди, добавление нового элемента возможно в **любом месте** списка, также можно удалить **любой** элемент списка. Ясно, что списковые структуры являются более гибкими, но и немного более сложными в реализации. Фактически, стеки и очереди можно считать частными случаями списков, в которых добавление и удаление элементов может выполняться только на концах.

Стандартный набор операций со списком включает:

* добавление нового элемента после заданного или перед заданным элементом с проверкой возможности добавления элемента
* удаление заданного элемента
* проход по списку от первого элемента к последнему с выполнением заданных действий
* поиск в списке заданного элемента

Как обычно, возможна статическая и динамическая реализация списков. При этом статическая реализация на базе массива может быть выполнена двумя способами. Будем использовать способ указателей.

Второй способ реализации списка на основе массива использует принцип **указателей** (но БЕЗ динамического распределения памяти). В этом случае каждый элемент списка (кроме последнего) должен содержать **номер** **ячейки** массива, в которой находится **следующий** за ним элемент. Это позволяет РАЗЛИЧАТЬ физический и логический порядок следования элементов в списке. Удобно (но НЕ обязательно) в начале массива ввести фиктивный элемент-заголовок, который всегда занимает нулевую ячейку массива, никогда не удаляется и указывает индекс первого реального элемента списка. В этом случае последний элемент списка (где бы он в массиве не располагался) должен в связующей части иметь некоторое специальное значение-признак, например – индекс 0.

Схема физического размещения элементов списка в массиве:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер ячейки | 0 | 1 | 2 | 3 | 4 | 5 |
| Информац. часть | заголовок | Элем. 3 | Элем. 1 |  | Элем. 2 | Элем. 4 |
| Следующий эл-нт | 2 | 5 | 4 |  | 1 | 0 |

Соответствующая схема логического следования элементов списка:

|  |
| --- |
| Элем. 1  (ячейка 2) |
| След.: 4 |

|  |
| --- |
| Элем. 2  (ячейка 4) |
| След.: 1 |

|  |
| --- |
| Элем. 4  (ячейка 5) |
| След.: 0 |

|  |
| --- |
| Элем. 3  (ячейка 1) |
| След.: 5 |

|  |
| --- |
| заголовок  (ячейка 0) |
| первый: 2 |

Тогда необходимые объявления могут быть следующими:

**Const  N = 100;**

**Type** TListItem = **record**

                                   Inf  :  <описание>;

                                   Next  :  **integer**;

**end**;

**Var**   StatList  :  **array** [0 . . N]  **of**   TListItem;

Рассмотрим реализацию основных списковых операций.

**Условие пустоты** списка:  StatList [ 0 ].Next = 0;

**Проход по списку:**

* ввести вспомогательную переменную Current для отслеживания текущего элемента списка и установить Current := StatList [ 0 ].Next;
* организовать цикл по условию  Current = 0, внутри которого обработать текущий элемент StatList [ Current ].Inf  и изменить указатель  Current  на следующий элемент:  Current := StatList [ Current ].Next

**Поиск элемента** аналогичен проходу, но может заканчиваться до достижения конца списка:

Current  :=  StatList [ 0 ].Next;

**While** (Current <> 0) **and** (StatList [ Current ].Inf  <> ‘значение’  **do**

      Current  :=  StatList [ Current ].Next;

**If**  Current = 0  **then**  ‘поиск неудачен’  **else**  ‘элемент найден’;

**Добавление** элемента **после** заданного:

* проверка возможности добавления с помощью счетчика текущего числа элементов в списке
* определение каким-то образом элемента, после которого надо добавить новый элемент (например – запрос у пользователя)
* поиск этого элемента в списке; пусть его индекс есть  i
* определение номера свободной ячейки массива для размещения нового элемента (методы определения будут рассмотрены ниже); пусть этот номер равен  j
* формирование  связующей части нового элемента, т.е. занесение туда номера ячейки из связующей части элемента  i : StatList [ j ].next  :=  StatList [ i ].next;
* изменение связующей части элемента  i  на номер j: StatList [ i ].next  :=  j;
* занесение данных в информационную часть нового элемента StatList[j].inf;

|  |
| --- |
| заданный элемент  (ячейка  i ) |
| next: след. элемент  (установить в  j ) |

|  |
| --- |
| следующий элемент |
| next: след. элемент |

|  |
| --- |
| новый элемент  (ячейка  j ) |
| next: из элемента  i |

Аналогично производится **добавление** нового элемента **перед** заданным, правда здесь приходится дополнительно узнавать номер ячейки, в которой находится элемент, **предшествующий** заданному. Это требует небольшого изменения процедуры поиска заданного элемента: вместе с индексом искомого элемента должен определяться индекс его предшественника. Для этого вводится вспомогательная переменная целого типа, которая в процессе поиска заданного элемента “отстает” на один элемент и тем самым всегда указывает предшественника искомого элемента.

**Алгоритм добавления** элемента перед заданным включает следующие шаги:

* проверка возможности добавления с помощью счетчика текущего числа элементов в списке
* определение каким-то образом элемента, перед которым надо добавить новый элемент (например – запрос у пользователя)
* поиск этого элемента в списке с одновременным отслеживанием элемента-предшественника; пусть индекс заданного элемента есть  i, а индекс его предшественника - s
* определение номера свободной ячейки массива для размещения нового элемента (методы определения будут рассмотрены ниже); пусть этот номер равен  j
* формирование  связующей части нового элемента, т.е. занесение туда индекса  i : StatList [ j ].next  :=  i;
* изменение связующей части элемента-предшественника с индекса  i  на индекс j: StatList [ s ].next  :=  j;
* занесение данных в информационную часть нового элемента StatList[j].inf;

|  |
| --- |
| предшеств. элемент  (ячейка   s ) |
| next: i  (установить в  j ) |

|  |
| --- |
| заданный элемент  (ячейка  i ) |
| next: след. элемент |

|  |
| --- |
| новый элемент  (ячейка  j ) |
| next: установить в  j |

**Удаление** заданного элемента (естественно, в случае его наличия в списке) также требует изменения связующей части у элемента-предшественника. Это изменение позволяет “обойти” удаляемый элемент и тем самым исключить его из списка.

Необходимые шаги:

* определение каким-то образом удаляемого элемента (например – запрос у пользователя)
* поиск удаляемого элемента в списке с одновременным отслеживанием элемента-предшественника; пусть индекс удаляемого элемента есть  i, а индекс его предшественника - s
* изменение связующей части элемента-предшественника с индекса  i  на индекс-значение связующей части удаляемого элемента  i: StatList [s].next  :=  StatList [ i ].next;
* обработка удаляемого элемента (например – вывод информационной части)
* включение удаленного элемента во вспомогательный список без его уничтожения или освобождение ячейки  i  с включением ее в список свободных ячеек (методы поддержки свободной памяти рассматриваются ниже)

|  |
| --- |
| следующий элемент  (например – j ) |
| next: (не изменяется) |

|  |
| --- |
| предшеств. элемент  (ячейка   s ) |
| next: i  (установить в  j ) |

|  |
| --- |
| удаляемый элемент  (ячейка   i ) |
| next: след. элемент  (например – j ) |

## 2.3 Краткое описание разработанных классов

При выполнении курсовой работы было создано 5 классов и одна главная подпрограмма с функцией main(). Через эту подпрограмму осуществляется диалог с пользователем Краткое описание разработанных классов (более подробно см. в пункте «Руководство программиста»):

**Класс Uchenik**. Является классом, содержащим структуру команды классов с учениками и методы взаимодействия с этой структурой (Get/Set методы).

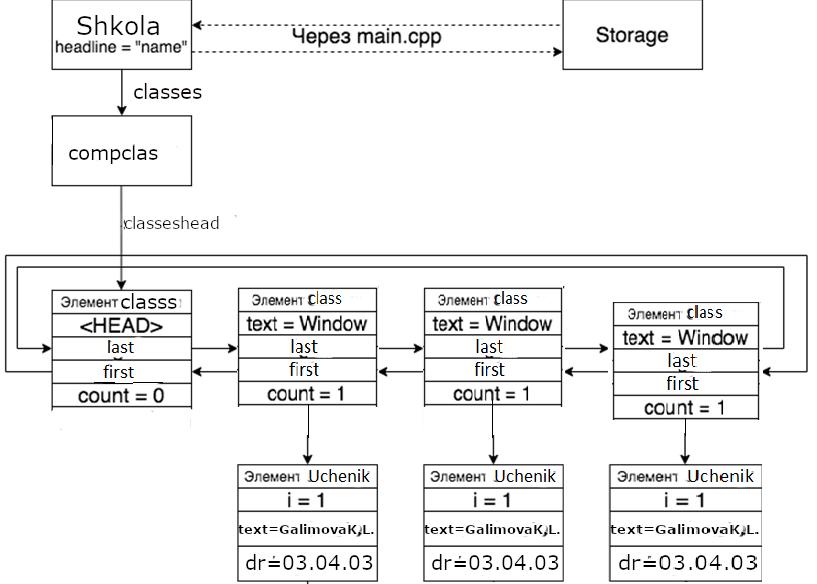
**Класс classes**. Является классом, содержащим структуру классов, методы взаимодействия с этой структурой-списком, реализованным на основе статического массива. Элементы списка - объекты класса Uchenik (класс classes является композицией команд).

**Класс Compclas**. Является композицией классов, то есть содержит в себе очередь, реализованную на основе статической распределения памяти. Элементами списка являются объекты структуры класса classes. Также класс Compclas содержит методы взаимодействия со списком и очередью, а именно: создание и удаление учеников и классов, их поиск по содержимому, конструктор и деструктор списка.

**Класс Shkola**. Данный класс осуществляет управление всеми классами, описанными ранее. Ведет диалог с главной программой, получает от нее команды и данные, выполняет на их основе необходимы действия. Содержит свое название headline и экземпляр класса **Compclas** (композиции классов) classs.

**Класс Storage**. Осуществляет взаимодействие меню и текстового файла. Получив ссылку на объект меню, считывает данные из файла и дополняет меню. Либо наоборот, сохраняет данные меню обратно в файл.

## 2.4 Схема структуры проекта (взаимодействия классов)



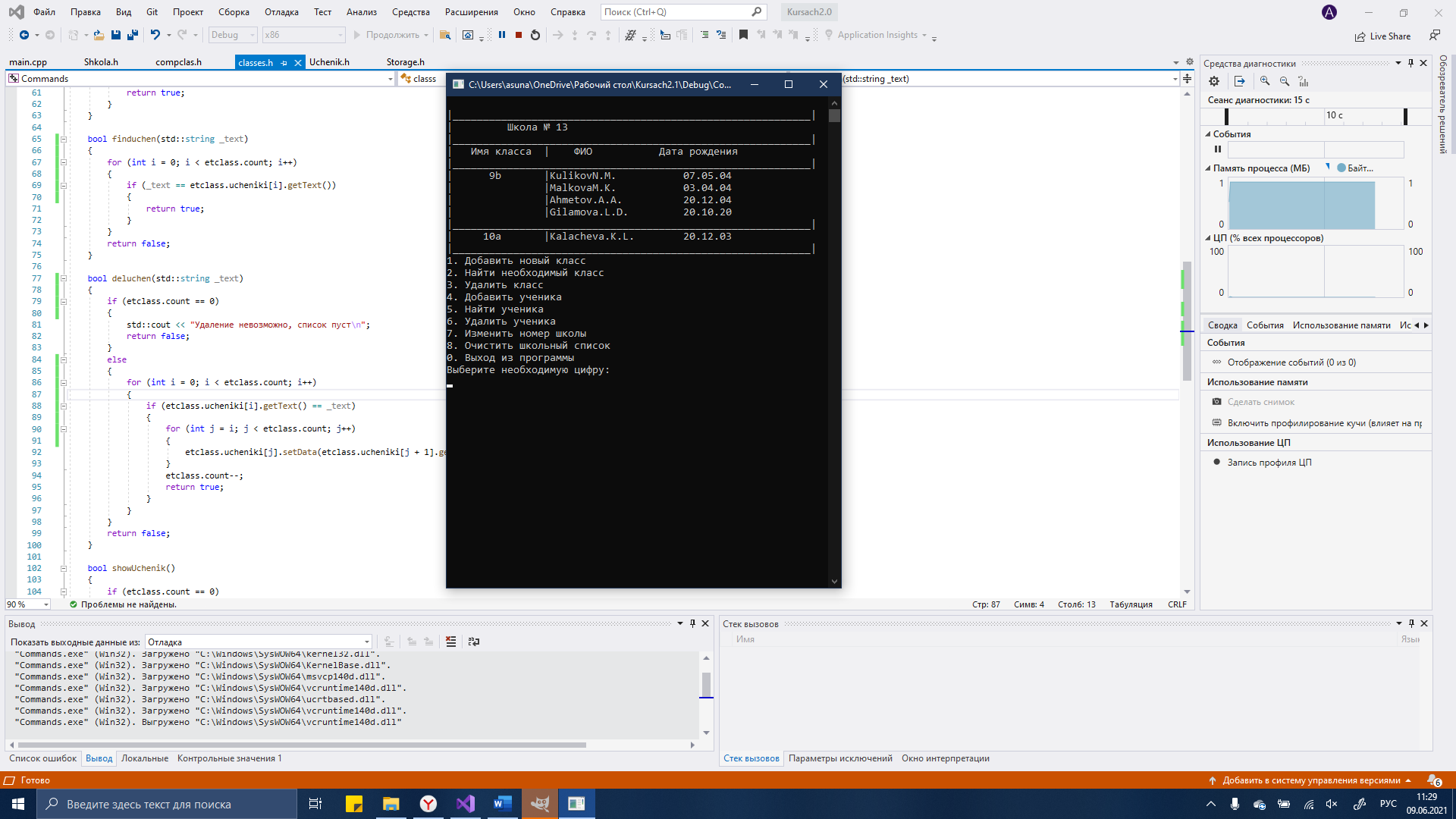
# 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа “Shkola” была разработана с целью демонстрации работы с комбинированными структурами данных.

В данной программе Вы можете создать информационный лист школы с информацией об классах и учениках в них. В класс вы можете добавить до десяти учеников. Информацией об ученике служит ФИО и дата рождения. Также программа поддерживает работу с файлом: при каждом запуске происходит поиск текстового файла “Kursov.txt” и ввод данных из него в таблицу (если такой файл существует и найден, иначе создается пустое меню)

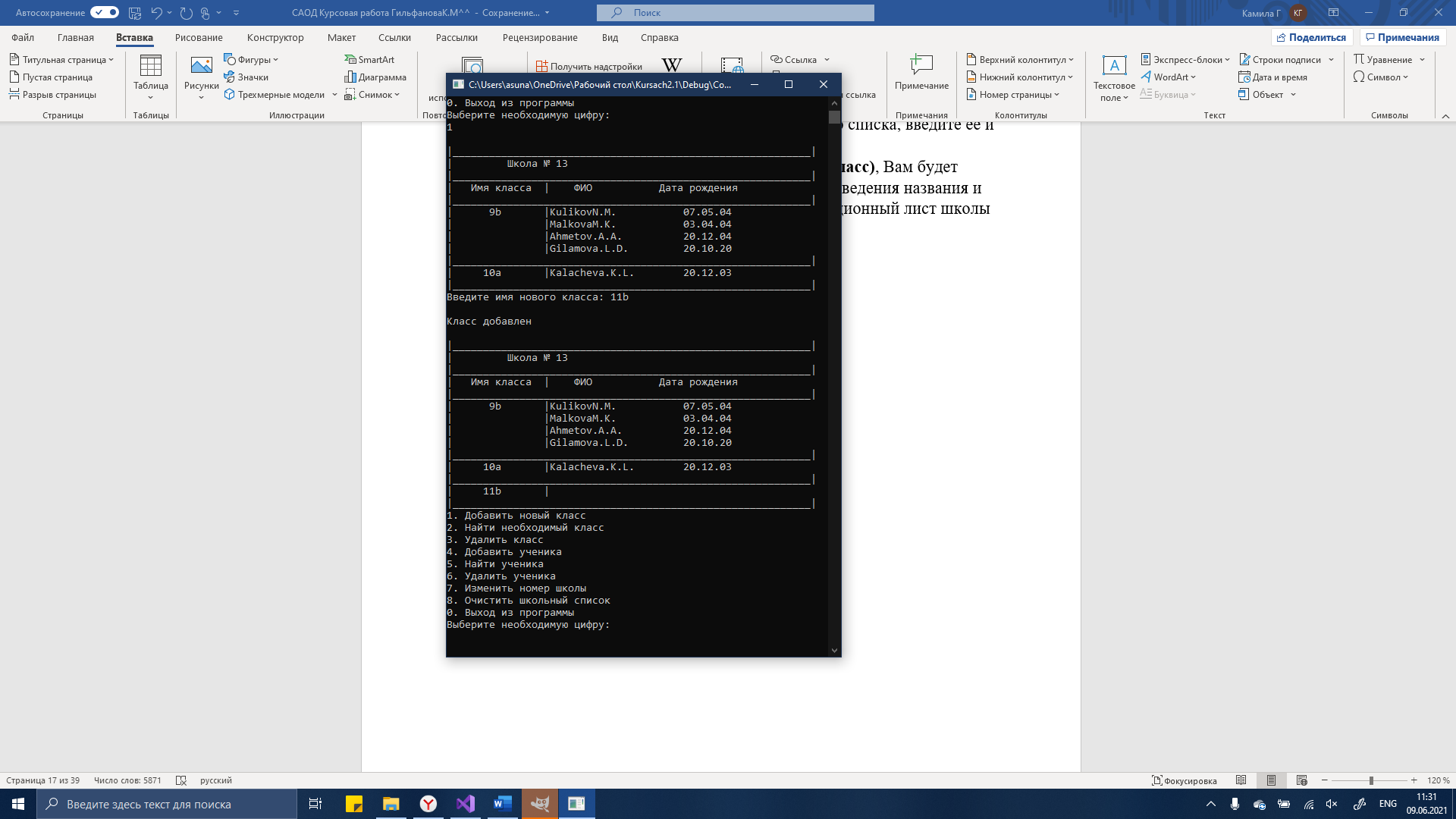
При завершении работы программы Вам будет предложено сохранить изменения в файле (сделать перезапись файла), но будьте осторожны: файл будет полностью перезаписан без возможности вернуть его в первоначальное состояние.

При запуске открывается информационный лист сохраненный в файле “Kursov.txt” в случае если файл не найден то будет предоставлен пустой информационный лист:

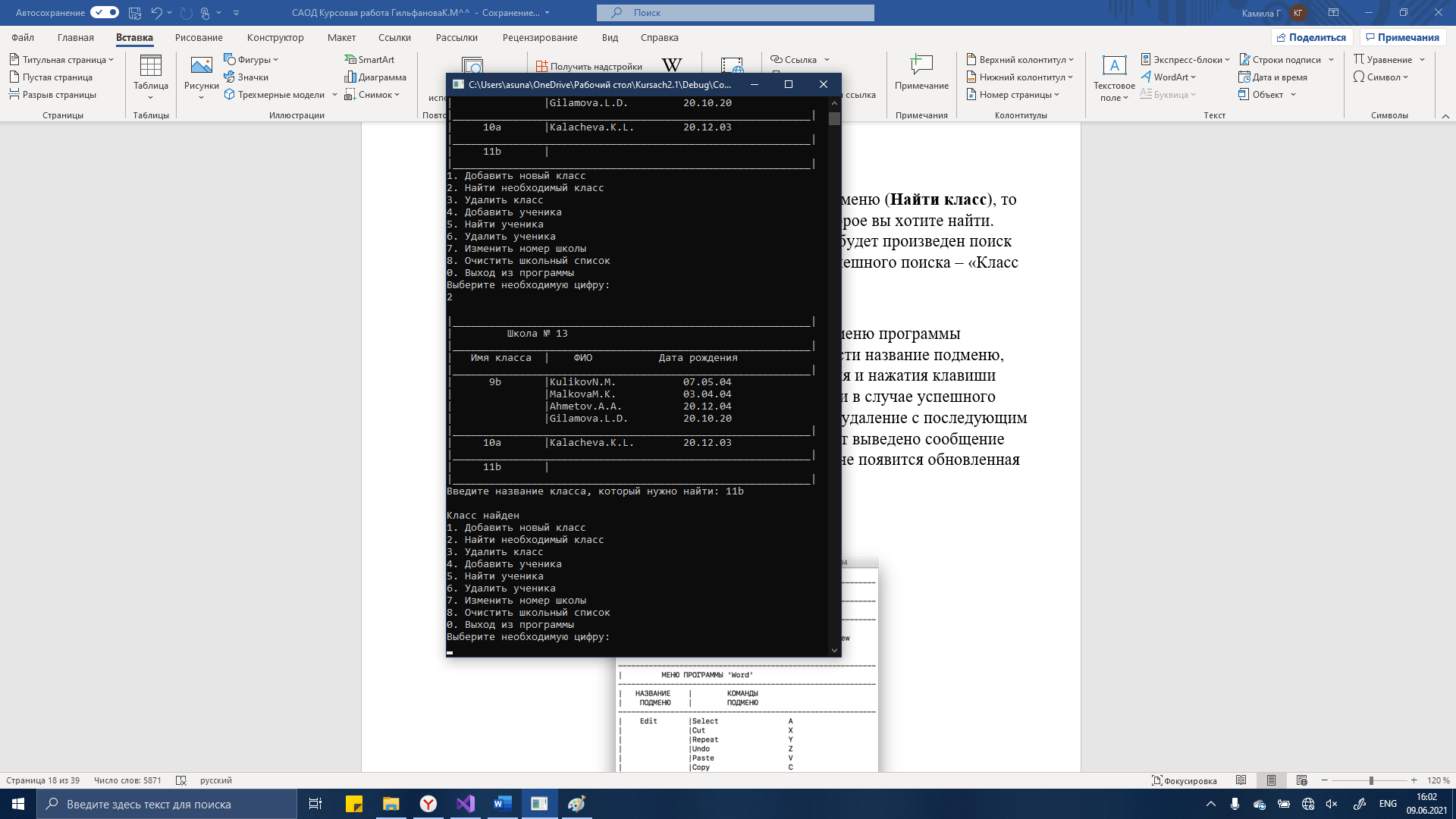


Выберите любую команду из предложенного списка, введите её и нажмите Enter

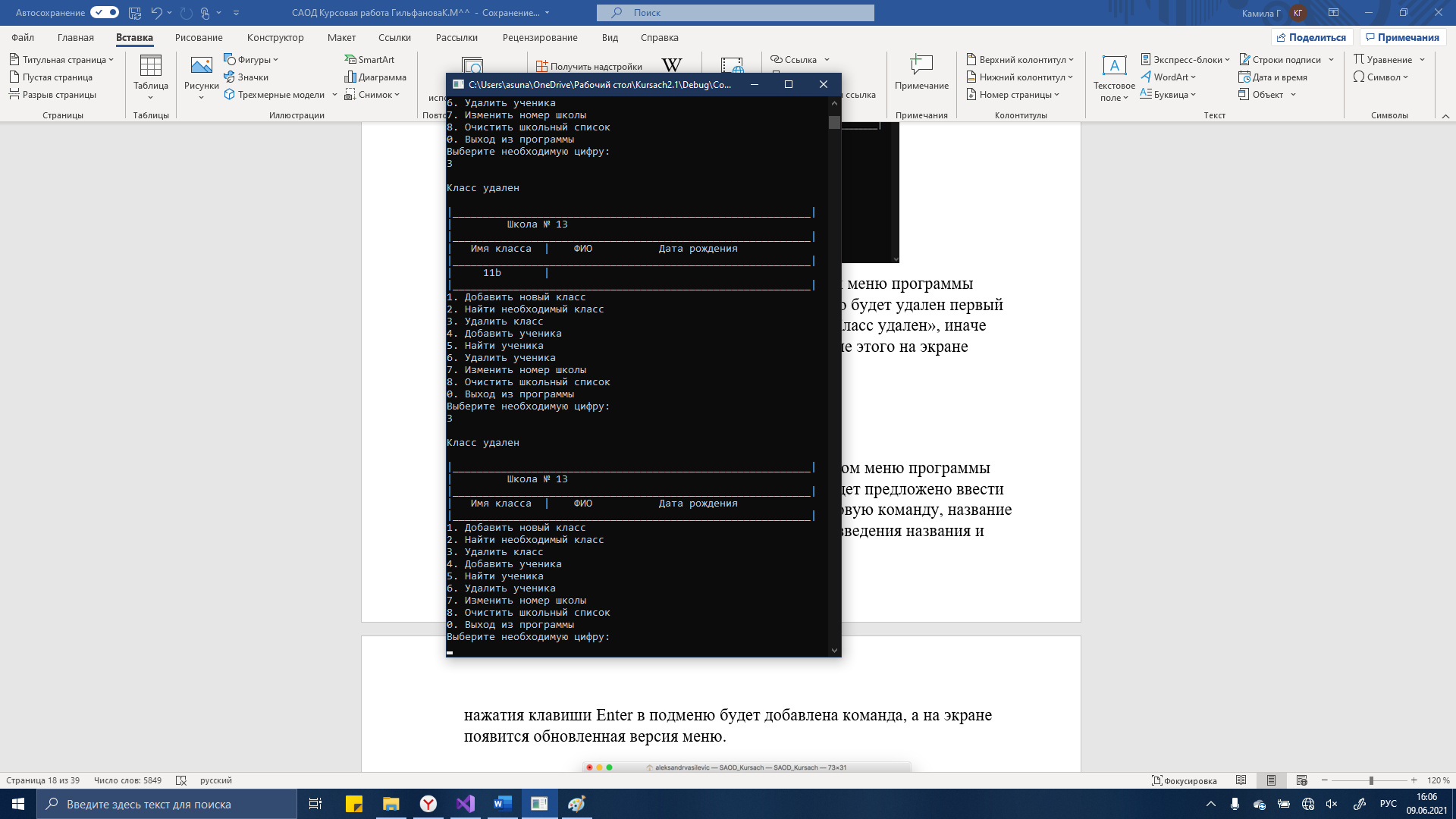
1.Выбрав **первый пункт (Добавить новый класс)**, Вам будет предложено ввести название нового класса. После введения названия и нажатия клавиши Enter класс добавится в информационный лист школы



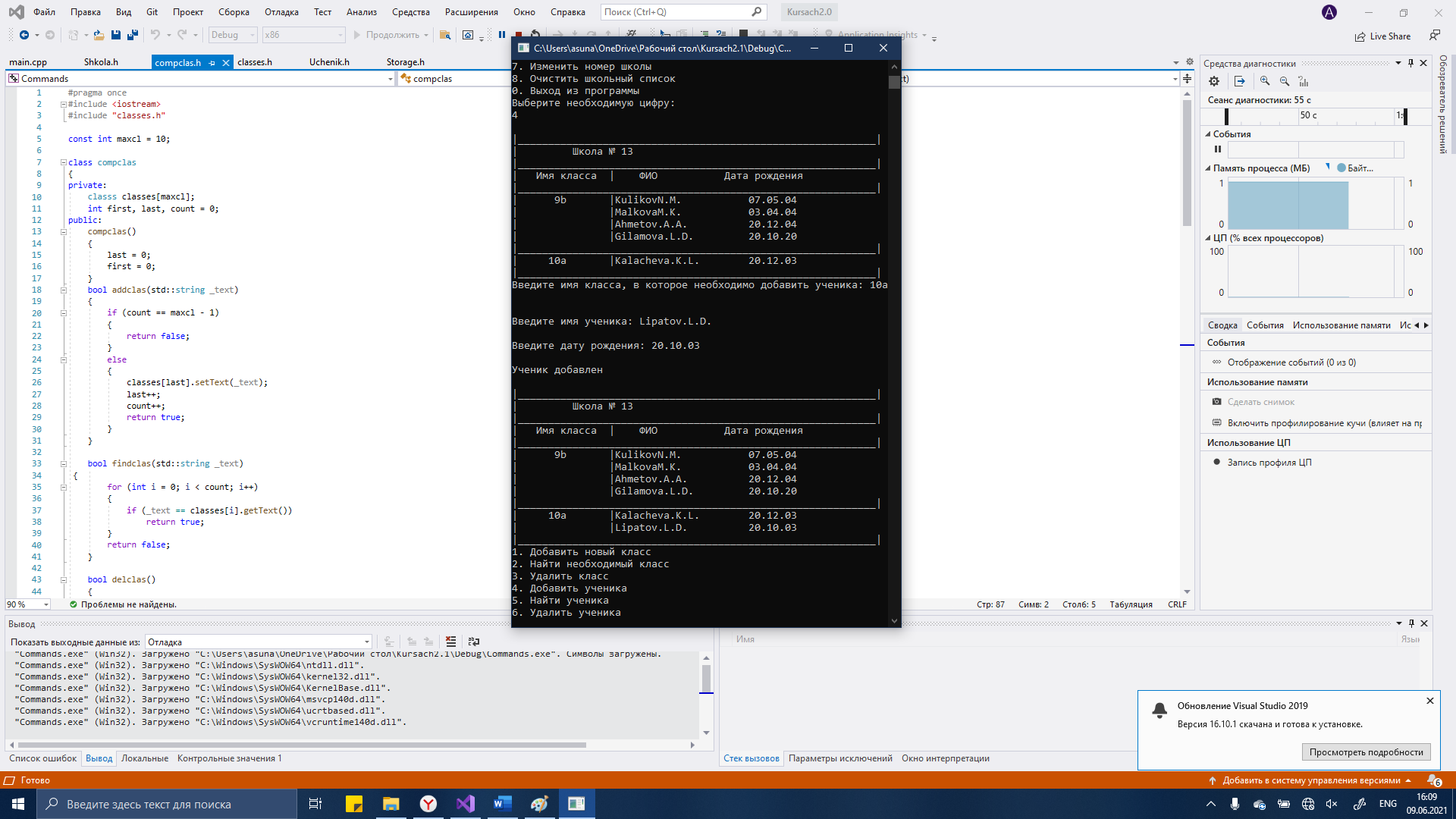
2.Если вы выберите **второй пункт** главного меню (**Найти класс**), то Вам будет предложено ввести название класса, которое вы хотите найти. После введения названия и нажатия клавиши Enter будет произведен поиск среди всех классов и вывод результата: в случае успешного поиска – «Класс найден», иначе – «Класс не найден»



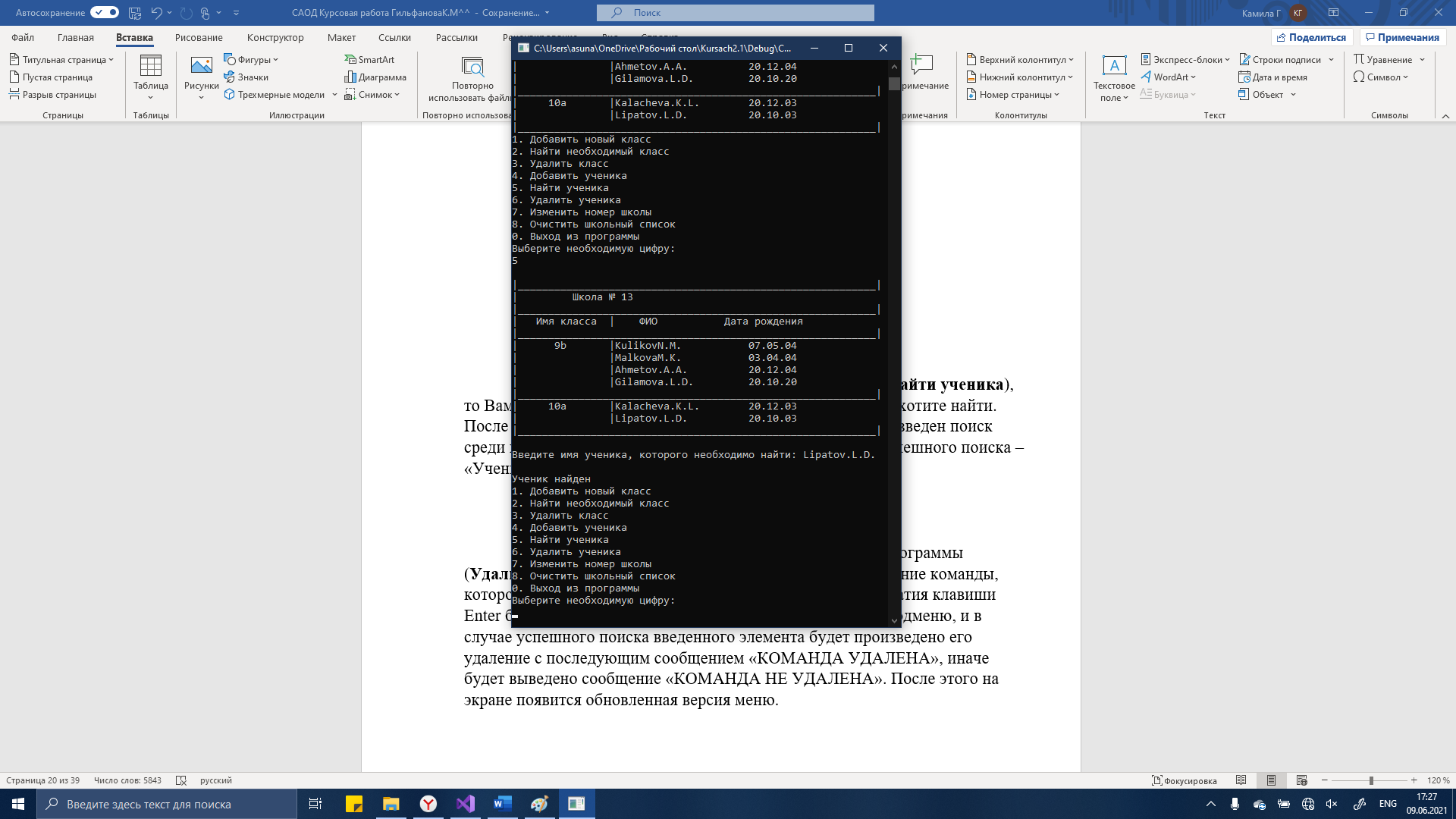
3.Если вы выберите **третий пункт** в главном менюпрограммы (**Удалить класс**) так как мы работаем с очередью то будет удален первый элемент из очереди с последующим сообщением «Класс удален», иначе будет выведено сообщение «Класс не удален». После этого на экране появится обновленная версия меню.



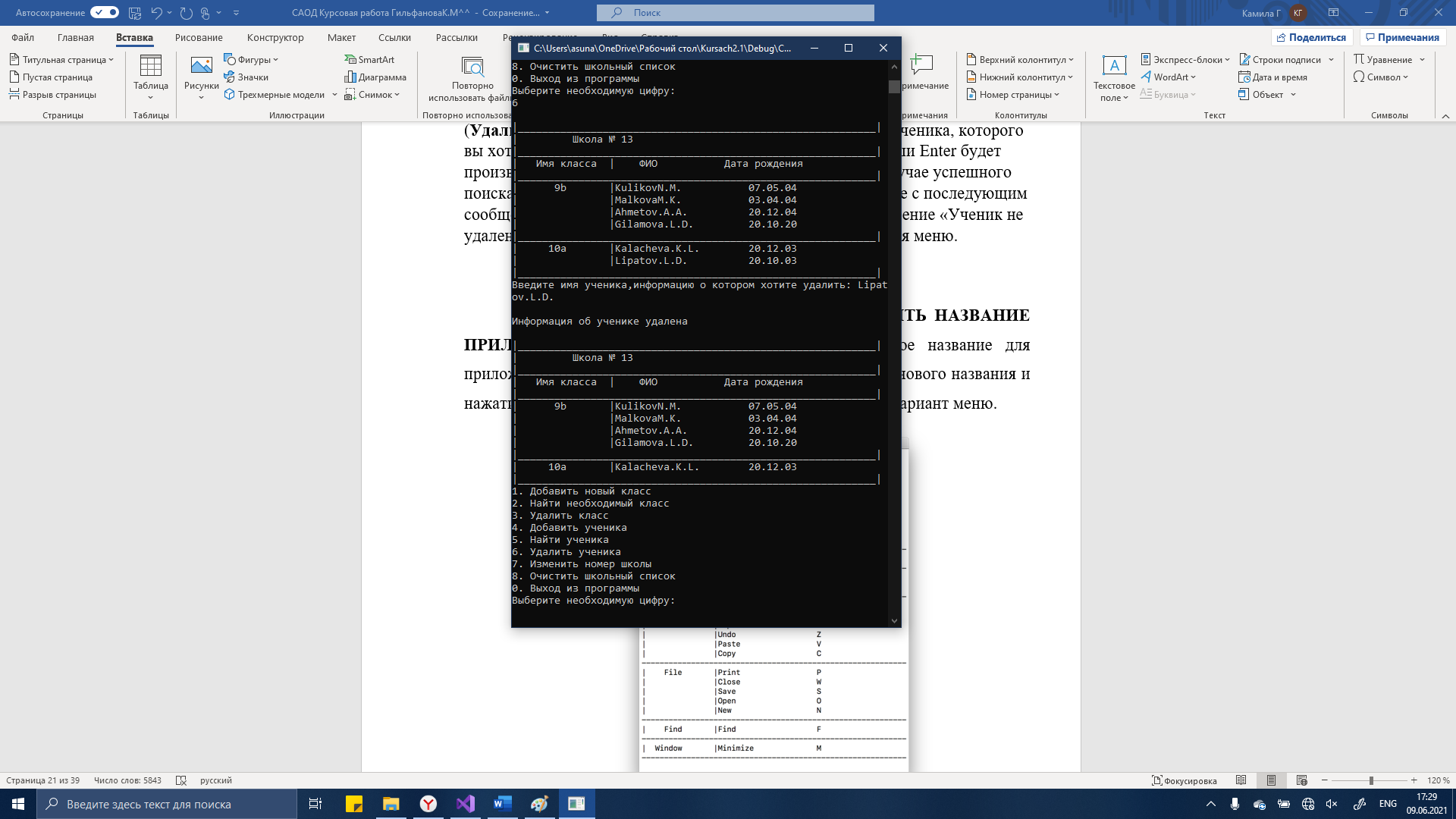
4.Если вы выберите **четвертый пункт** в главном менюпрограммы (**Добавить нового ученика**), то Вам будет предложено ввести класс, в который вы хотите добавить нового ученика, имя ученика и дата рождения. После введения названия и нажатия клавиши Enter в класс будет добавлен ученик, а на экране появится обновленная версия меню.



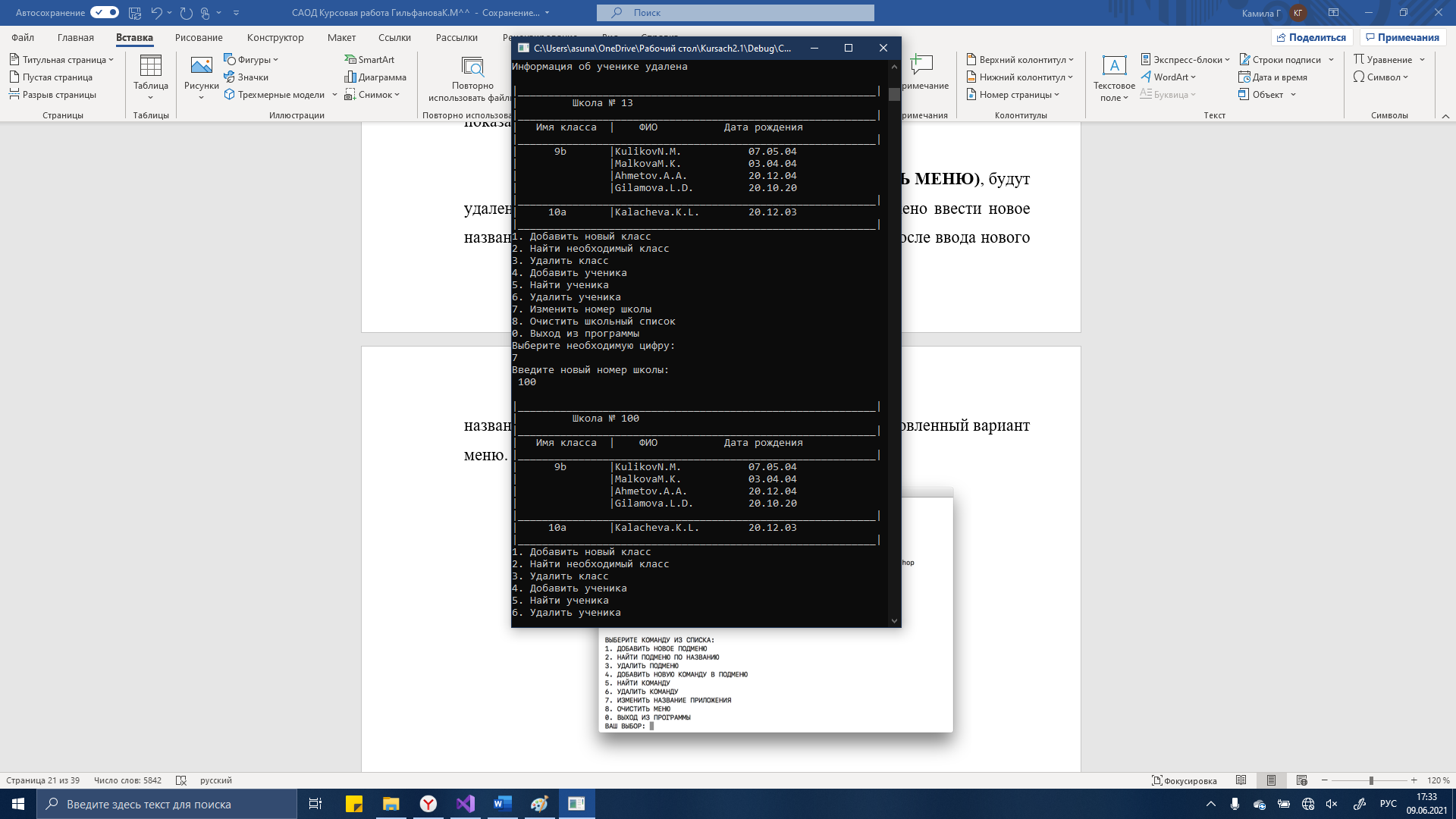
5.Если вы выберите **пятый пункт** главного меню (**Найти ученика**), то Вам будет предложено ввести имя ученика, которого вы хотите найти. После введения имени и нажатия клавиши Enter будет произведен поиск среди имен каждого класса и вывод результата: в случае успешного поиска – «Ученик найден», иначе – «Ученик не найден»



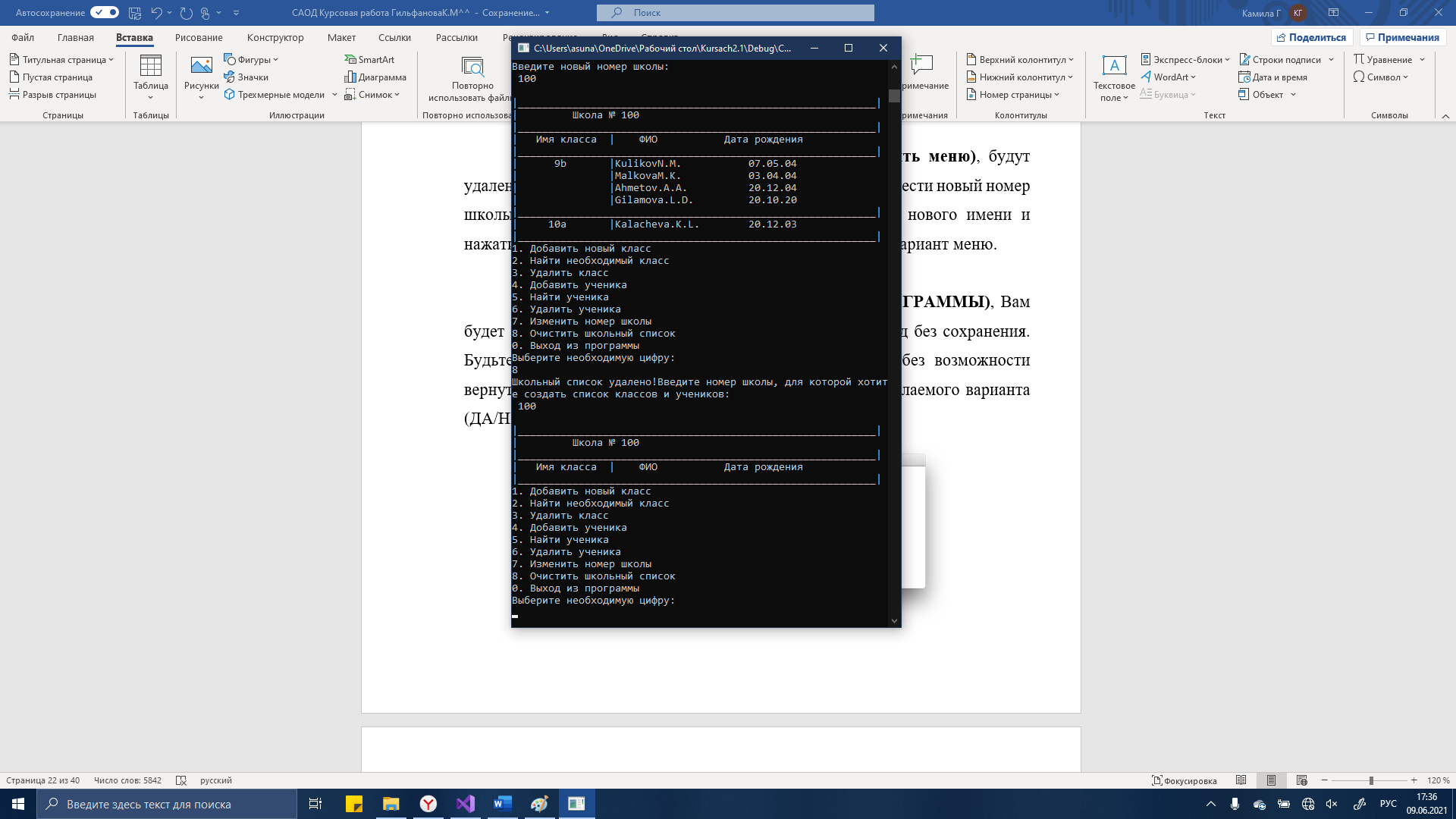
6.Если вы выберите **шестой пункт** в главном менюпрограммы (**Удалить ученика**), то Вам будет предложено ввести имя ученика, которого вы хотите удалить. После введения имени и нажатия клавиши Enter будет произведен поиск среди всех команд каждого класса, и в случае успешного поиска введенного элемента будет произведено его удаление с последующим сообщением «Ученик удален», иначе будет выведено сообщение «Ученик не удален». После этого на экране появится обновленная версия меню.



7.Выбрав **седьмой пункт** главного меню **(Изменить номер школы)**, Вам будет предложено ввести новый номер школы, для которого вы создаете меню. После ввода нового номера и нажатия клавиши «Enter» вам будет показан обновленный вариант меню.

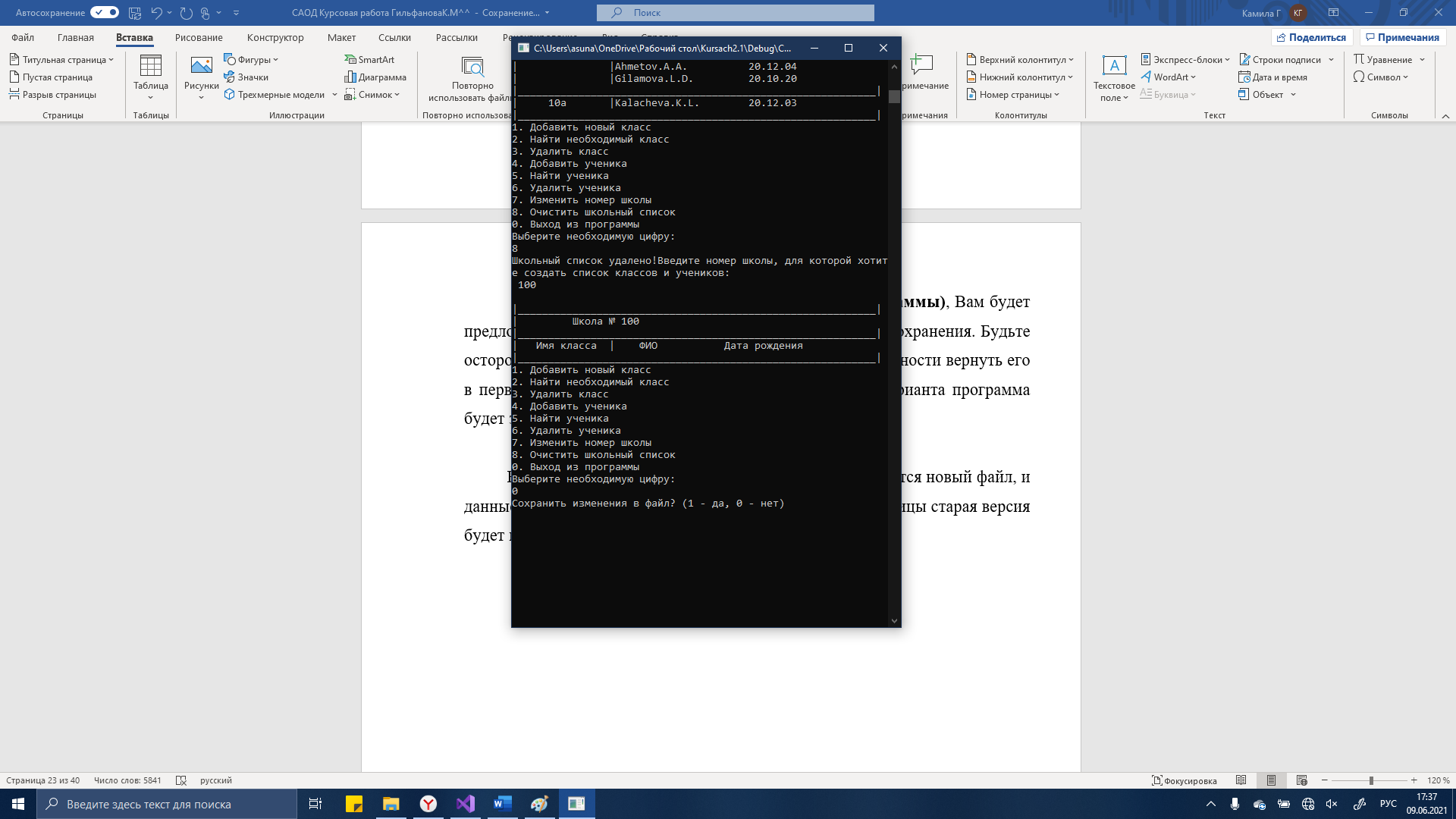


8.Выбрав **восьмой** **пункт** главного меню **(Очистить меню)**, будут удалены все классы с учениками, и Вам будет предложено ввести новый номер школы, для которого будет создано меню. После ввода нового имени и нажатия клавиши «Enter» вам будет показан обновленный вариант меню.



9.Выбрав **пункт 0** главного меню **(Выход из программы)**, Вам будет предложено сохранить изменения в файле или выход без сохранения. Будьте осторожны: файл будет полностью перезаписан без возможности вернуть его в первоначальное состояние. После выбора желаемого варианта программа будет завершена.

Если при сохранении файл не будет найден, то создастся новый файл, и данные будут сохранены. Осторожно: при сохранении таблицы старая версия будет полностью утеряна!



# 4. РУКОВОДСТВО ПРОГРАММИСТА

Данный проект был создан для демонстрации работы комбинированной структуры «Статическая очередь неупорядоченных статических списков» с использованием объектно-ориентированного программирования (ООП). Для создания программы был выбран язык C++ с поддержкой ООП.

Главный структура – статическая очередь на статической распределении памяти. Подструктура – неупорядоченный список на статическом распределении памяти на основе массива.

В ходе разработки было создано 5 классов, каждый из которых был помещен в отдельный файл. Также был создан отдельный файл с главной подпрограммой, содержащей функцию «int main()». С полными алгоритмами каждого из классов можно ознакомиться в приложении к пояснительной записке. Для взаимодействия с таблицей меню главная программа обращается только к объекту класса Shkola.h, а тот уже взаимодействует со всеми остальными классами.

Описание методов классов:

**Класс Uchenik.** Является классом, содержащим структуру команды классов и методы взаимодействия с ней.

Структура “UchenikData etuchenik ” содержит две переменные: “string text” – ФИО ученика и “string dr” –дата рождения ученика.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Uchenik() | Конструктор пустой команды, все поля структуры получают значения по умолчанию |
| void Set(std::string \_text, std::string \_dr) | Метод ввода данных в структуру объекта |
| std::string GetText() | Возвращает номер поля “text” из структуры команды |
| std::string Getdr() | Возвращает номер поля “dr” из структуры команды |
| void Clear() | Метод очистки структуры. Присваиваются значения по умолчанию |
| ~ Command () | Деструктор объекта класса Command |

**Класс classes.** Является классом, содержащим структуру классов школы, методы взаимодействия с ним, а также неупорядоченный список, реализованный на основе статического массива, элементами являются объекты класса Uchenik (то есть класс classes является композицией учеников).

Структура “classData etclass” содержит следующие переменные:

string text – название данного класса

uchenik ucheniki [maxuch] – массив учеников(для статического неупорядоченного списка)

int count – счетчик количества учеников

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| classes() | Конструктор объекта класса Submenu (подменю). Все поля структуры принимают значения по умолчанию, в классе создается информация об ученике. |
| classes (std::string \_text) | Конструктор объекта класса. Отличается от прошлого тем, что поле принимает переданное значение. |
| void setText(std::string \_text) | Set метод для получения имени ученика |
| std::string getText() | Get метод для поля text структуры объекта |
| bool Adduchen (string \_text, char \_dr) | Метод добавления ученика. Добавляет элемент после или до выбранного элемента |
| int Finduchen(string \_text) | Метод поиска ученика с заданным содержимым. Если поиск прошел неудачно, то вернется false, иначе – true |
| bool deluchen (string \_text) | Удаляет ученика с заданным содержимым (если её поиск прошел успешно) |
| void ShowUchenik() | Выводит на экран название класс и учеников |
| string GetString() | Формирует текстовую строку, содержащую название текущего класса и учеников, хранящихся в нем. Метод необходим для записи данных в файл |
| ~ classes () | Деструктор объекта класса classes |

**Класс Compclas.** Является композицией классов, содержит в себе очередь, реализованный на основе статической распределении памяти. Элементами списка являются объекты класса classes.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Compclas () | Конструктор объекта класса Compclas. При его вызове создается элемент композии классов |
| classs classes[maxcl]; | Массив для хранения имен классов |
| int first, last, count = 0; | Переменные счетчика , последней и первой переменной |
| Bool addclas (std::string \_text) | Добавление нового класса |
| classes \*findclas (std::string \_text) | Поиск класса по названию. Возвращает ссылку на класс |
| bool delclas (std::string name) | Удаление класса с заданным названием |
| bool adduchen (std::string \_name, std::string \_text, std::string \_dr) | Добавление ученика в заданное подменю, в конец списка |
| int finduchen (std::string \_text) | Поиск ученика по содержимому |
| bool deluchen (std::string \_text) | Удаление ученика по содержимому |
| void Showclasses() | Метод вывода композиции классов на экран |
| string GetString() | Метод формирования текстовой строки для записи в файл |
| ~ Compclas () | Деструктор объекта класса Compclas |

**Класс Shkola.** Данный класс осуществляет управление всеми ранее описанными классами. Ведет диалог с главной программой main.cpp, получает от нее коман

ды и данные, выполняет на их основе необходимы действия. Содержит свойство - название приложения string nomer и экземпляр класса композиции подменю Compclas Classes.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Shkola(std::string \_headline) | Конструктор экземпляра класса таблицы.  Присваивает таблице номер школы. |
| Shkola () | Второй конструктор этого класса, устанавливает названию приложения по умолчанию – "15" |
| void SetHead(std::string \_head) | Set-метод для номера школы |
| string GetHead() | Get-метод для номера школы |
| bool addclas (std::string \_text) | Добавление нового класса. Вызывает одноименный метод объекта classes |
| bool findclas (std::string \_text) | Поиск класса по названию |
| bool delclas (std::string name) | Удаление класса с заданным названием. Вызывает метод delclas объекта Classes. |
| bool adduchen (std::string name, std::string \_text, string \_dr) | Добавление ученика в заданный класс, в список. Вызывает соответствующий метод adduchen объекта Classes |
| int finduchen(std::string \_text) | Поиск ученика по её содержимому |
| bool deluchen (std::string \_text) | Удалить кченика с заданным содержимым  Вызывает соответствующий метод deluchen объекта Classes |
| void show() | Вывод всего меню на экран |
| string GetString() | Формирование меню в виде строки для дальнейшей записи в файл |
| ~Shkola() | Деструктор объекта класса Shkola |

**Класс Storage.** Осуществляет взаимодействие меню и текстового файла. Получив ссылку на объект школы, считывает данные из файла и дополняет школу. Либо наоборот, сохраняет данные меню обратно в файл. Будьте осторожны: файл будет полностью перезаписан без возможности вернуть его в первоначальное состояние.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Storage(string \_address) | конструктор, получает адрес файла и сохраняет его в виде строки для взаимодействия в будущем. |
| bool InputData(Shkola \*\_menu) | метод извлечения данных из файла и заполнения ими меню |
| bool OutputData(Shkola \*\_menu) | метод извлечения данных из меню с последующим внесением их в файл |
| ~Storage() | деструктор объекта класса Storage |

## ВЫВОД

В результате выполнения данной курсовой работы была создана программа, реализующая структуру данных Статическая очередь неупорядоченных статических списков*,*  используя принципы объектно-ориентированного программирования. Были повторены и закреплены знания, полученные в ходе изучения курсов «Структуры и алгоритмы обработки данных» и «Объектно-ориентированное программирование», а также были выработаны новые навыки владения языком программирования С++.

На основе исходных данных выполнены все требования к реализации программы, получен ожидаемый результат.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Козин А. Н. Структуры и алгоритмы обработки данных. Учебно-методическое пособие. Издательство: Татарский институт содействия бизнесу, 2003.

2. Козин А. Н. Объектно-ориентированное программирование. Учебно-методическое пособие. Издательство: Академия Управления «ТИСБИ», 2006.

# ПРИЛОЖЕНИЕ. ЛИСТИНГИ ВСЕХ ФАЙЛОВ ПРОЕКТА

main.cpp – главная программа

#include <iostream>

#include <windows.h>

#include "Shkola.h"

#include "Storage.h"

Shkola\* menu;

Storage file("C:/Users/asuna/OneDrive/Рабочий стол/Kursach2.1/Kursov.txt");

int Check()

{

int num;

while (!(std::cin >> num))

{

std::cin.clear();

while (std::cin.get() != '\n');

std::cout << "Ошибка ввода\n";

}

return num;

}

void addclas()

{

std::string name;

menu->show();

std::cout << "Введите имя нового класса: ";

std::cin >> name;

if (menu->addclas(name))

{

std::cout << "\nКласс добавлен\n ";

}

else {

std::cout << "\nКласс не добавлен (очередь заполнена)\n";

}

menu->show();

}

void findclas()

{

std::string name;

menu->show();

std::cout << "Введите название класса, который нужно найти: ";

std::cin >> name;

if (menu->findclas(name))

std::cout << "\nКласс найден\n";

else

std::cout << "\nКласс не найден\n";

}

void delclas()

{

if (menu->delclas())

{

std::cout << "\nКласс удален\n";

menu->show();

}

else

std::cout << "\nНечего удалять\n";

}

void adduchen()

{

std::string name, uchenik;

std::string dr;

menu->show();

std::cout << "Введите имя класса, в которое необходимо добавить ученика: ";

std::cin >> name;

if (menu->findclas(name))

{

std::cout << "\nВведите имя ученика: ";

std::cin >> uchenik;

std::cout << "\nВведите дату рождения: ";

std::cin >> dr;

if (menu->adduchen(name, uchenik, dr))

{

std::cout << "\nУченик добавлен\n";

menu->show();

}

}

else

std::cout << "\nТакого класса не существует\n";

}

void finduchen()

{

std::string \_text;

menu->show();

std::cout << "\nВведите имя ученика, которого необходимо найти: ";

std::cin >> \_text;

if (menu->finduchen(\_text))

std::cout << "\nУченик найден\n";

else

std::cout << "\nУченик не найден\n";

}

void deluchen()

{

std::string \_text;

menu->show();

std::cout << "Введите имя ученика,информацию о котором хотите удалить: ";

std::cin >> \_text;

if (menu->deluchen(\_text))

{

std::cout << "\nИнформация об ученике удалена\n";

menu->show();

}

}

void mainMenu()

{

menu = new Shkola();

std::string headline = "";

if (!file.InputData(menu))

{

std::cout << "Файл не найден\n";

std::cout << "Введите номер школы:\n ";

std::cin >> headline;

menu->setHead(headline);

}

menu->show();

int n;

while (true)

{

std::cout << "1. Добавить новый класс\n";

std::cout << "2. Найти необходимый класс\n";

std::cout << "3. Удалить класс\n";

std::cout << "4. Добавить ученика\n";

std::cout << "5. Найти ученика\n";

std::cout << "6. Удалить ученика\n";

std::cout << "7. Изменить номер школы\n";

std::cout << "8. Очистить школьный список\n";

std::cout << "0. Выход из программы\n";

std::cout << "Выберите необходимую цифру:\n";

n = Check();

while (n < 0 || n > 8)

{

std::cout << "Ошибка ввода\nВыберите необходимую цифру:\n";

n = Check();

}

if (n == 1)

addclas();

else if (n == 2)

findclas();

else if (n == 3)

delclas();

else if (n == 4)

adduchen();

else if (n == 5)

finduchen();

else if (n == 6)

deluchen();

else if (n == 7)

{

std::cout << "Введите новый номер школы:\n ";

std::cin >> headline;

menu->setHead(headline);

menu->show();

}

else if (n == 8)

{

delete menu;

std::cout << "Школьный список удалено!";

std::cout << "Введите номер школы, для которой хотите создать список классов и учеников:\n ";

std::cin >> headline;

menu = new Shkola(headline);

menu->show();

}

else if (n == 0) {

std::cout << "Сохранить изменения в файл? (1 - да, 0 - нет)\n";

n = Check();

while ((n < 0) || (n > 1)) {

std::cout << "Ошибка ввода\n Введите вновь: ";

n = Check();

}

std::cout << "\n";

if (n == 1) {

if (file.OutputData(menu))

std::cout << "Данные успешно записаны в файл\n";

else

std::cout << "Ошибка сохранения в файл\n";

}

break;

}

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "Russian");

mainMenu();

}

Storage.h – класс для работы с файлом (чтение и запись)

#pragma once

#pragma warning(disable:4996)

#include "Shkola.h"

#include <fstream>

#include <iostream>

class Storage

{

private:

std::string address;

public:

Storage(std::string \_address)

{

address = \_address;

};

bool InputData(Shkola\* \_menu)

{

std::string text, name,dr;

char buffer[100];

char\* str;

std::ifstream file(address);

if (!file.is\_open())

return false;

if (!file.eof())

{

file.getline(buffer, 100);

text = buffer;

\_menu->setHead(text);

}

else

{

std::cout << "Файл пуст\n";

return false;

}

while (!file.eof())

{

file.getline(buffer, 100);

str = strtok(buffer, " ");

if (str == NULL)

continue;

if (str)

name = str;

\_menu->addclas(name);

str = strtok(NULL, " ");

while (str)

{

if (str)

text = str;

str = strtok(NULL, " ");

if ((str))

dr = str;

else

return false;

\_menu->adduchen(name, text, dr);

str = strtok(NULL, " ");

}

}

file.close();

return true;

}

bool OutputData(Shkola\* \_menu) {

std::ofstream file(address);

if (!file.is\_open())

return false;

file << \_menu->GetString();

file.close();

return true;

};

~Storage() { };

};

Shkola.h – класс для взаимодействия с меню (композицией классов)

#pragma once

#include <iostream>

#include "compclas.h"

class Shkola {

private:

compclas Classes;

std::string nomer;

public:

Shkola()

{

nomer = "15";

}

Shkola(std::string \_head)

{

nomer = \_head;

}

void setHead(std::string \_head)

{

nomer = \_head;

}

std::string getHead()

{

return nomer;

}

bool addclas(std::string \_text)

{

return Classes.addclas(\_text);

}

bool findclas(std::string \_text)

{

return Classes.findclas(\_text);

}

bool delclas()

{

return Classes.delclas();

}

bool adduchen(std::string \_name, std::string \_text, std::string \_dr)

{

return Classes.adduchen(\_name, \_text, \_dr);

}

bool finduchen(std::string \_text)

{

return Classes.finduchen(\_text);

}

bool deluchen(std::string \_text)

{

return Classes.deluchen(\_text);

}

void show() {

std::cout << "\n|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

std::cout << "| Школа № " << nomer << "";

std::cout << "\n|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|";

std::cout << "\n| Имя класса |" << " ФИО Дата рождения \n";

Classes.showclasses();

std::cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

}

std::string GetString() {

return nomer + '\n' + Classes.GetString();

};

~Shkola() {};

};

Compclas.h – класс композиции классов

#pragma once

#include <iostream>

#include "classes.h"

const int maxcl = 10;

class compclas

{

private:

classs classes[maxcl];

int first, last, count = 0;

public:

compclas()

{

last = 0;

first = 0;

}

bool addclas(std::string \_text)

{

if (count == maxcl - 1)

{

return false;

}

else

{

classes[last].setText(\_text);

last++;

count++;

return true;

}

}

bool findclas(std::string \_text)

{

for (int i = 0; i < count; i++)

{

if (\_text == classes[i].getText())

return true;

}

return false;

}

bool delclas()

{

if (count == 0)

{

return false;

}

else {

for (int i = 0; i < last; i++)

{

classes[i] = classes[i + 1];

}

last--;

count--;

return true;

}

}

bool adduchen(std::string \_name, std::string \_text, std::string \_dr)

{

int i;

for (i = 0; i < count; i++)

{

if (\_name == classes[i].getText())

{

break;

}

}

if (classes[i].adduchen(\_text, \_dr))

{

return true;

}

return false;

}

bool finduchen(std::string \_text)

{

for (int i = 0; i < count; i++)

{

if (classes[i].finduchen(\_text))

return true;

}

return false;

}

bool deluchen(std::string \_text)

{

for (int i = 0; i < count; i++)

{

if (classes[i].deluchen(\_text))

return true;

}

return false;

}

void showclasses()

{

if (count == 0)

{

return;

}

for (int i = 0; i < count; i++)

{

std::cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

std::cout << "|" << std::setw(8) << classes[i].getText() << std::setw(8) << "|";

if (!(classes[i].showUchenik()))

std::cout << "\n";

}

}

std::string GetString()

{

std::string str = "";

for (int i = 0; i < count; i++)

{

std::stringstream ss;

ss << classes[i].getText();

str += ss.str();

str += classes[i].GetString();

str += '\n';

}

return str;

}

~compclas() {}

};

classes.h – класс со структурой классов(композиции учеников)

#pragma once

#include <iostream>

#include <iomanip>

#include <sstream>

#include "Uchenik.h"

const int maxuch = 10;

class classs

{

private:

struct classsData

{

std::string text;

uchenik ucheniki[maxuch];

int count = 0;

}etclass;

public:

classs()

{

etclass.ucheniki[0].setData("<ПУСТО>", " ");

etclass.text = "<ПУСТО>";

}

classs(std::string \_text)

{

etclass.ucheniki[0].setData("<ПУСТО>", " ");

etclass.text = \_text;

}

void setText(std::string \_text)

{

etclass.text = \_text;

}

std::string getText()

{

return etclass.text;

}

bool adduchen(std::string \_text, std::string \_dr)

{

bool flag = true;

int i=1;

if (etclass.count >= maxuch - 1)

{

std::cout << "Добавление невозможно, список заполнен\n";

return false;

}

else {

if (etclass.ucheniki[0].getText() == "<ПУСТО>")

{

etclass.ucheniki[0].setData(\_text, \_dr);

}

else

{

etclass.ucheniki[etclass.count].setData(\_text, \_dr);

}

etclass.count++;

return true;

}

}

bool finduchen(std::string \_text)

{

for (int i = 0; i < etclass.count; i++)

{

if (\_text == etclass.ucheniki[i].getText())

{

return true;

}

}

return false;

}

bool deluchen(std::string \_text)

{

if (etclass.count == 0)

{

std::cout << "Удаление невозможно, список пуст\n";

return false;

}

else

{

for (int i = 0; i < etclass.count; i++)

{

if (etclass.ucheniki[i].getText() == \_text)

{

for (int j = i; j < etclass.count; j++)

{

etclass.ucheniki[j].setData(etclass.ucheniki[j + 1].getText(), etclass.ucheniki[j + 1].getdr());

}

etclass.count--;

return true;

}

}

}

return false;

}

bool showUchenik()

{

if (etclass.count == 0)

{

return false;

}

else {

for (int i = 0; i < etclass.count; i++)

{

if (i >= 1)

std::cout << "|" << std::setw(16) << std::right << "|";

std::cout << std::setw(20) << std::left << etclass.ucheniki[i].getText() << " " << std::right << etclass.ucheniki[i].getdr() << "\n";

}

}

return true;

}

std::string GetString()

{

std::string str = "";

for (int i = 0; i < etclass.count; i++)

{

std::stringstream ss;

ss << etclass.ucheniki[i].getText();

str += " ";

str += ss.str();

str += " ";

str += etclass.ucheniki[i].getdr();

}

return str;

};

~classs() {};

};

Uchenik.h – класс со структурой учеников класса

#pragma once

#include <iostream>

class uchenik

{

private:

struct uchenikData

{

std::string text;

std::string dr;

}etuchenik;

public:

uchenik()

{

etuchenik.text = "<ПУСТО>";

etuchenik.dr = ' ';

}

uchenik(std::string \_text, std::string \_dr)

{

etuchenik.text = \_text;

etuchenik.dr = \_dr;

}

void setData(std::string \_text, std::string \_dr)

{

etuchenik.text = \_text;

etuchenik.dr = \_dr;

}

std::string getText()

{

return etuchenik.text;

}

std::string getdr()

{

return etuchenik.dr;

}

~uchenik() {}

};