СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 3

АННОТАЦИЯ 5

ВВЕДЕНИЕ 6

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7

1.1 Структурное программирование 7

1.2 Процедурное программирование 7

1.3 Модульное программирование 7

1.4 Ссылки 8

1.5 Работа с памятью. Указатели 8

1.6 Работа с файлами 8

1.7 Динамические структуры данных 9

1.8 Линейный список 9

1.9 Функции 9

1.10 Объектно-ориентированное программирование 10

1.11 Конструкторы и деструкторы 11

1.12 Классы и дружественные функции 11

1.12.1 Описание классов и их методов 12

1.13 Методы шифрования и дешифрования 13

2 ПРАКТИЧЕСКАЯ ЧАСТЬ 17

2.1 Структура программы 17

2.2 Алгоритм решения задачи 20

2.3 Программная реализация задания 21

2.3.1 Пример выполнения задания 21

2.3.2 Отображение списка студентов 25

Для отображения студентов используется operator<< в классе Cursework 25

2.3.3 Изменение данных о студенте 26

Данные о студенте изменяются по принципу замещения старого объекта студента в базе данных новым. Для этого сначала находится искомый нами студент с помощью метода find, после этого, сохранив его копию, мы выполняем изменения в этой копии, после изменений, с помощью метода append мы удаляем студента из базы данных, а после удаляем с помощью remove. 26

2.3.3 Добавление информации о новом студенте 26

Студент в базу данных добавляется с помощью метода append: создается объект класса Student, а после он передается в этот метод в качестве аргумента. 26

2.3.4 Удаление данных о студенте 27

Удаление данных о студенте происходит с помощью метода remove. 27

2.3.5 Шифрование данных 28

Шифрование данных производится с помощью RSA и AES128. Генерируется случайный ключ, с помощью него шифруется файл алгоритмом AES128, а потом шифруется и сам ключ с помощью RSA. 28

2.4 Руководство пользователя 29

2.5 Системные требования 29

ЗАКЛЮЧЕНИЕ 30

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 31

ПРИЛОЖЕНИЕ А Исходный код программы 32

Main.cpp 32

Student.h 35

Student.cpp 36

List.h 37

tolkachev.h 43

tolkachev.cpp 44

АННОТАЦИЯ

Тема курсовой работы: Разработка программы информационного поиска студентов по заданным критериям с возможностью шифрования данных

Выполнил: Иванов Глеб Максимович, БББО-07-23

Работа состоит из введения, двух глав, заключения и списка используемых источников.

Во введении рассмотрены цели, задачи работы и ее актуальность.

В первой главе, относящейся к теоретической части, рассмотрены и приведены основные термины, понятия и определения из языка программирования С++.

Во второй главе, относящейся к практической части, приведены листинги кода, представлена программа, выполняющая поставленную задачу, а приведены блок схемы и разобраны алгоритмы ее работы.

В заключении приведены основные выводы, полученные в хотя выполнения работы.

Ключевые слова: С++, объектно-ориентированное программирование, шифрование и дешифрование, шифрование данных, проектирование базы информационного поиска студентов, поиск по заданным критериям.

ВВЕДЕНИЕ

Задачей курсовой работы является:

Вариант 79. Отсортировать всю группу по увеличению года

поступления в ВУЗ, с указанием интервала года рождения.

Цель курсовой работы по дисциплине «Языки программирования» состоит в закреплении и углублении знаний и навыков, полученных при изучении дисциплины. Курсовая работа предполагает выполнение задания повышенной сложности по проектированию, разработке и тестированию программного обеспечения, а также оформлению сопутствующей документации.

В результате изучения дисциплины обучающийся должен:

Знать:

1) языки, системы и инструментальные средства программирования в профессиональной деятельности;

2) математический аппарат, математически пакеты, программные комплексы;

3) общие принципы построения и использования современных языков программирования высокого уровня.

Уметь:

1) использовать языки, системы и инструментальные средства программирования в профессиональной деятельности;

2) строить алгоритм решения задачи, проводить его анализ и реализовывать в современных программных комплексах;

3) работать с интегрированной средой разработки программного обеспечения.

Владеть:

1) языками программирования, системами и инструментальными средствами программирования в профессиональной деятельности;

2) навыками разработки, документирования, тестирования и отладки программ на языке программирования высокого уровня;

3) основными методами разработки алгоритмов и программ;

4) методами создания структур данных, используемые для представления типовых информационных объектов.

Задачи курсовой работы:

1) проанализировать исходные данные, указанные в задании;

2) определить данные, структуры, классы, методы и функции, необходимые для выполнения работы согласно варианту;

3) разработать соответствующей алгоритм решения конкретной задачи;

4) реализовать элементы, описанные в пункте 2;

5) подготовить контрольные данные для тестирования программного обеспечения;

6) отладить разработанное программное обеспечение на основе контрольных данных, подготовленных в предыдущем пункте.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Структурное программирование

Структурное программирование воплощает принципы системного подхода в процессе создания и эксплуатации программного обеспечения ЭВМ. В основу структурного программирования положены следующие достаточно простые положения:

1) алгоритм и программа должны составляться поэтапно;

2) сложная задача должна разбиваться на достаточно простые части, каждая из которых имеет один вход и один выход;

3) логика алгоритма и программы должна опираться на минимальное число достаточно простых базовых управляющих структур.

Принципы структурного программирования:

1) программа построена из трёх базовых управляющих конструкций: последовательность, ветвление, цикл;

2) в программе базовые управляющие конструкции вложены друг в друга произвольным образом;

3) повторяющиеся фрагменты программы были оформлены в виде подпрограмм (процедур и функций);

4) перечисленные конструкции должны имеют один вход и один выход;

5) разработка программы ведётся пошагово, методом «сверху вниз».

1.2 Процедурное программирование

Процедурное программирование — программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка. Процедурное программирование является отражением архитектуры традиционных ЭВМ.

Особенности процедурного программирования:

1) предопределенные функции. Предопределенная функция — это инструкция, идентифицируемая именем. Обычно предопределенные функции встроены в языки программирования более высокого уровня, но они получены из библиотеки или реестра, а не из программы;

2) локальная переменная. Локальная переменная — это переменная, которая объявлена, в основной структуре метода и ограничена локальной областью действия, которую она задает. Локальная переменная может использоваться только в том методе, в котором она определена;

3) глобальная переменная. Глобальная переменная — это переменная, которая объявляется вне любой другой функции, определенной в коде. Благодаря этому глобальные переменные могут использоваться во всех функциях, в отличие от локальной переменной;

4) модульность: Модульность — это когда две разные системы имеют под рукой две разные задачи, но сгруппированы вместе, чтобы сначала выполнить более крупную задачу. В этом случае каждая группа систем будет выполнять свои собственные задачи один за другим, пока все задачи не будут выполнены;

5) передача параметров: Передача параметров — это механизм, используемый для передачи параметров в функции, подпрограммы или процедуры.

1.3 Модульное программирование

Модуль в программировании — это фрагмент кода, имеющий определенное функциональное значение и характеризующийся логической завершенностью.

Модульное программирование — это способ создания программы посредством объединения модулей в единую структуру.

В основе модульного программирования лежат три основных концепции:

Основные концепции модульного программирования:

1) каждый модуль имеет единственную точку входа и выхода;

2) размер модуля по возможности должен быть минимизирован;

3) вся система построена из модулей;

4) каждый модуль не зависит от того, как реализованы другие модули.

Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

1.4 Ссылки

Ссылка — это псевдоним для другой переменной. Они объявляются при помощи символа &. Ссылки должны быть проинициализированы при объявлении, причем только один раз.

Ссылка при определении сразу же инициализируется. Инициализация ссылки производится следующим образом:

int i = 0;

int& iref = i;

Физически iref представляет собой постоянный указатель на int - переменную типа int\* const.

Поскольку ссылка — это псевдоним, то при передаче объекта в функцию по ссылке внутри нее объект можно изменять. Ссылки не могут ссылаться на другие ссылки или на поле битов. Не может быть массивов ссылок или указателей на ссылку. Ссылка может использоваться для возврата результата из функции. Возвратить результат по ссылке — значит возвратить не указатель на объект и не его значение, а сам этот объект.

1.5 Работа с памятью. Указатели

Указатели являются одним из основных понятий языка Си. В такие переменные можно записывать адреса любых участков памяти, на чаще всего – адрес начального элемента динамического массива.

Важно знать, что:

1) указатель – это переменная, в которой записан адрес другой переменной;

2) при объявлении указателя надо указать тип переменных, на которых он будет указывать, а перед именем поставить знак \*;

3) знак & перед именем переменной обозначает ее адрес;

4) знак \* перед указателем в рабочей части программы (не в объявлении) обозначает значение ячейки, на которую указывает указатель;

5) нельзя записывать по указателю, который указывает непонятно куда – это вызывает сбой программы, поскольку что-то стирается в памяти;

6) для обозначения недействительного указателя используется константа NULL;

7) при изменении значения указателя на n он в самом деле сдвигается к n-ому следующему числу данного типа, то есть для указателей на целые числа на n\*sizeof(int) байт;

8) указатель печатаются по формату %p.

1.6 Работа с файлами

Файл – это именованная область ячеек памяти, в которой хранятся данные одного типа. Файл имеет следующие характерные особенности: уникальное имя; однотипность данных; произвольная длина, которая ограничивается только емкостью диска. Для работы с файлом в языке C++ необходима ссылка на файл. Для определения такой ссылки существует структура FILE, описанная в заголовочном файле stdio.h. Данная структура содержит все необходимые поля для управления файлами, например, текущий указатель буфера, текущий счетчик байтов, базовый адрес буфера ввода-вывода, номер файла.

Функция открытия файла.

При открытии файла (потока) в программу возвращается указатель на поток (файловый указатель), являющийся указателем на объект структурного типа FILE. Этот указатель идентифицирует поток во всех последующих операциях.

Функция закрытия файла.

Открытые на диске файлы после окончания работы с ними рекомендуется закрыть явно. Это является хорошим тоном в программировании.

Функция переименования файла.

Функция переименовывает файл; первый параметр – старое имя файла, второй – новое. Возвращает 0 при неудачном выполнении.

Функция контроля конца файла

Для контроля достижения конца файла есть функция feof. int feof(FILE \* filename);

1.7 Динамические структуры данных

Часто в серьезных программах надо использовать данные, размер и структура которых должны меняться в процессе работы. Динамические массивы здесь не выручают, поскольку заранее нельзя сказать, сколько памяти надо выделить – это выясняется только в процессе работы. Например, надо проанализировать текст и определить, какие слова и в каком количество в нем встречаются, причем эти слова нужно расставить по алфавиту. В таких случаях применяют данные особой структуры, которые представляют собой отдельные элементы, связанные с помощью ссылок.

1.8 Линейный список

В простейшем случае каждый узел содержит всего одну ссылку. Для определенности будем считать, что решается задача частотного анализа текста – определения всех слов, встречающихся в тексте и их количества. В этом случае область данных элемента включает строку (длиной не более 40 символов) и целое число.

1.9 Функции

Функция определяет действия, которые выполняет программа. Функции позволяют выделить набор инструкций и придать ему имя. А затем многократно по присвоенному имени вызывать в различных частях программы. По сути функция — это именованный блок кода.

Функцию можно вызвать из любого количества мест в программе. Значения, передаваемые функции, являются аргументами, типы которых должны быть совместимы с типами параметров в определении функции.

Длина функции практически не ограничена, однако для максимальной эффективности кода целесообразно использовать функции, каждая из которых выполняет одиночную, четко определенную задачу. Сложные алгоритмы лучше разбивать на более короткие и простые для понимания функции, если это возможно. Функции могут быть перегружены, что означает, что разные версии функции могут использовать одно и то же имя, если они отличаются числом и типом формальных параметров.

Определение функции состоит из объявления и тела функции, заключенных в фигурные скобки, которые содержат объявления переменных, операторы и выражения. В следующем примере показано полное определение функции.

1.10 Объектно-ориентированное программирование

Объектно-ориентированное программирование – это подход, при котором вся программа рассматривается как набор взаимодействующих друг с другом объектов. При этом нам важно знать их характеристики.

Основные задачи ООП — структурировать код, повысить его читабельность и ускорить понимание логики программы. Косвенно выполняются и другие задачи: например, повышается безопасность кода и сокращается его дублирование.

Такой подход помогает строить сложные системы более просто и естественно благодаря тому, что вся предметная область разбивается на объекты и каждый из них слабо связан с другими объектами. Слабая связанность возникает вследствие соблюдения трех принципов: инкапсуляции, наследования и полиморфизма.

1) инкапсуляция – сокрытие поведения объекта внутри него. Объекту «водитель» не нужно знать, что происходит в объекте «машина», чтобы она ехала. Это ключевой принцип ООП;

2) наследование. Есть объекты «человек» и «водитель». У них есть явно что-то общее. Наследование позволяет выделить это общее в один объект (в данном случае более общим будет человек), а водителя — определить, как человека, но с дополнительными свойствами и/или поведением. Например, у водителя есть водительские права, а у человека их может не быть;

3) полиморфизм – это переопределение поведения. Можно снова рассмотреть «человека» и «водителя», но теперь добавить «пешехода». Человек умеет как-то передвигаться, но как именно, зависит от того, водитель он или пешеход. То есть у пешехода и водителя схожее поведение, но реализованное по-разному: один перемещается ногами, другой – на машине.

ООП позволяет упростить сложные объекты, составляя их из более маленьких и простых, поэтому над программой могут работать сотни разработчиков, каждый из которых занят своим блоком. Большинство современных языков программирования — объектно-ориентированные, и, однажды поняв суть, вы сможете освоить сразу несколько языков.

1.11 Конструкторы и деструкторы

Конструктор – это функция-член, имя которой совпадает с именем класса, инициализирующая переменные-члены, распределяющая память для их хранения.

Деструктор – это функция-член, имя которой представляет собой имя класса, предназначенная для уничтожения переменных.

При создании объектов одной из наиболее широко используемых операций которую вы будете выполнять в ваших программах, является инициализация элементов данных объекта. Чтобы упростить процесс инициализации элементов данных класса, C++ использует специальную функцию, называемую конструктором, которая запускается для каждого создаваемого вами объекта. Также C++ обеспечивает функцию, называемую деструктором, которая запускается при уничтожении объекта.

Конструктор представляет собой метод класса, который облегчает вашим программам инициализацию полей при создании объекта класса. Конструктор имеет такое же имя, как и сам класс. Конструктор не имеет возвращаемого значения. Конструкторы относят к интерфейсу класса, чтобы с их помощью можно было создавать объекты данного класса из внешней части программы.

Таким образом, деструктор не может быть перегружен и должен существовать в классе в единственном экземпляре. Деструктор вызывается автоматически при уничтожении объекта.

1.12 Классы и дружественные функции

Класс представляет составной тип, который может использовать другие типы. Классы и объекты в С++ являются основными концепциями объектно-ориентированного программирования — ООП.

Класс предназначен для описания некоторого типа объектов. То есть класс является планом объекта. А объект представляет конкретное воплощение класса, его реализацию. Объекты — конкретное представление абстракции, которые имеют свои свойства и методы. Свойства — это любые данные, которыми можно характеризовать объект класса. Методы — это функции, выполняющие различные действия над данными (свойствами) класса. Поле класса в объектно-ориентированном программировании — переменная, описание которой создает программист при создании класса. Все данные объекта хранятся в его полях.

Класс может определять переменные и константы для хранения состояния объекта и функции для определения поведения объекта.

На объекты классов, как и на объекты других типов, можно определять указатели. Затем через указатель можно обращаться к членам класса - переменным и методам. Однако если при обращении через обычную переменную используется символ точка, то для для обащения к членам класса через указатель применяется стрелка (->).

Дружественные функции — это функции, которые не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private.

Для определения дружественных функций используется ключевое слово friend.

1.12.1 Описание классов и их методов

Классы в языке программировании С++ — это абстракция, которая описывает методы и свойства, ещё не существующих объектов. Объекты — конкретное представление абстракции, которые имеют свои свойства и методы. Свойства — это любые данные, которыми можно характеризовать объект класса. Методы — это функции, выполняющие различные действия над данными (свойствами) класса. Поле класса в объектно-ориентированном программировании — переменная, описание которой создает программист при создании класса. Все данные объекта хранятся в его полях.

Для разработки данного проекта понадобятся следующие классы. В таблице ниже представлены и описаны поля и методы класса, используемые в проекте.

template <typename T> struct Node {

T unit;

unsigned int index;

Node<T>\* next;

};

template <typename T> class List {

private:

Node<T>\* first;

Node<T>\* last;

public:

List ();

List (const List& unit);

List& operator= (const List& unit);

T& operator[] (const unsigned int index);

bool operator== (List& unit) const;

~List ();

void append (const T newData);

void remove (const unsigned int index);

unsigned int find (T data);

unsigned int length ();

bool isExists (T data);

};

struct grade {

std::string subject;

std::string date;

short mark;

};

class Student {

private:

std::string number;

std::string first\_name;

std::string middle\_name;

std::string last\_name;

unsigned short year;

std::string birthday;

std::string faculty;

std::string department;

std::string group;

std::string gender;

List <grade> grades;

public:

Student () {};

Student (const Student& unit);

Student (

const std::string number, const std::string first\_name, const std::string middle\_name, const std::string last\_name,

const unsigned short year, const std::string birthday, const std::string faculty, const std::string department,

const std::string group, const std::string gender, const List <grade> grades

);

Student& operator= (const Student& unit);

bool operator== (const Student& unit1) const;

std::string getNumber () const { return this->number; }

std::string getFirstName () const { return this->first\_name; }

std::string getMiddleName () const { return this->middle\_name; }

std::string getLastName () const { return this->last\_name; }

unsigned short getYear () const { return this->year; }

std::string getBirthday () const { return this->birthday; }

std::string getFaculty () const { return this->faculty; }

std::string getDepartment () const { return this->department; }

std::string getGroup () const { return this->group; }

std::string getGender () const { return this->gender; }

List <grade> getGrades () const { return this->grades; }

void setNumber (const std::string data) { this->number = data; }

void setFirstName (const std::string data) { this->first\_name = data; }

void setMiddleName (const std::string data) { this->middle\_name = data; }

void setLastName (const std::string data) { this->last\_name = data; }

void setYear (const unsigned short data) { this->year = data; }

void setBirthday (const std::string data) { this->birthday = data; }

void setFaculty (const std::string data) { this->faculty = data; }

void setDepartment (const std::string data) { this->department = data; }

void setGroup (const std::string data) { this->group = data; }

void setGender (const std::string data) { this->gender = data; }

void appendGrade (grade data) { this->grades.append(data); }

}

class Exercise79 {

private:

List <Student> database;

public:

Exercise79 () {}

Exercise79 (const Exercise79& unit);

Exercise79 (const std::string filename);

Exercise79& operator= (const Exercise79& unit);

friend std::ostream& operator<< (std::ostream& os, Exercise79& unit);

~Exercise79 () {}

void toFile (const std::string filename);

void append (const Student unit);

void remove (const unsigned int index);

unsigned int find (const Student unit);

Student get (const unsigned int index);

bool isExists (const Student unit);

List <Student> sortGroup (std::string group);

void sortList (List <Student>& unit) const;

static void encrypt (const std::string filenameIn, const std::string filenameOut);

static void decrypt (const std::string filenameIn, const std::string filenameOut);

};

1.13 Методы шифрования и дешифрования

Шифрование – это процесс кодирования информации с целью предотвращения несанкционированного доступа.

Дешифрование – процесс, обратный процессу шифрованию.

OpenSSL – криптографическая библиотека с открытым исходным кодом. Библиотека поддерживает почти все низкоуровневые алгоритмы хеширования и шифрования, а также реализует большинство популярных криптографических стандартов, в том числе: позволяет создавать ключи RSA, DH, DSA, шифровать данные.

Для начала реализуем функцию Crypt, для этого поэтапно реализуем следующее:

1) генерацию рандомного пароля;

2) шифрование алгоритмом AES базы данных;

3) удаление не зашифрованной базы данных;

4) запись пароля в файл;

5) шифрование алгоритмом RSA файла с паролем с помощью публичного ключа;

6) удаление не зашифрованного файла с паролем.

Листинг 1.13.1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | srand(time(NULL));  char\* pass = new char[64];  for (int i = 0; i < 64; ++i) {  switch (rand() % 3) {  case 0:  pass[i] = rand() % 10 + '0';  break;  case 1:  pass[i] = rand() % 26 + 'A';  break;  case 2:  pass[i] = rand() % 26 + 'a';  }  } |

Листинг 1.13.2

|  |  |
| --- | --- |
| 1  -  2  3 | string command = "openssl\\bin\\openssl.exe enc -aes-256-cbc -salt -in file.txt -out file.txt.enc -pass pass:";  command += pass;  system(command.c\_str()); |

Листинг 1.13.3

|  |  |
| --- | --- |
| 1  2  3 | if (remove("file.txt") != 0) {  cout << "[ERROR] - deleting file" << endl;  } |

Листинг 1.13.4

|  |  |
| --- | --- |
| 1  2  3  4 | ofstream file;  file.open("key.txt", ios::binary);  file.write(pass, 65);  file.close(); |

Листинг 1.13.5

|  |  |
| --- | --- |
| 1  -  2 | command = "openssl\\bin\\openssl.exe rsautl -encrypt -inkey rsa.public -pubin -in key.txt -out key.txt.enc";  system(command.c\_str()); |

Листинг 1.13.6

|  |  |
| --- | --- |
| 1  2  3 | if (remove("key.txt") != 0) {  cout << "[ERROR] - deleting file" << endl;  } |

Для реализации функции Decrypt поэтапно реализуем следующее:

1) Расшифровка зашифрованного файла с паролем с помощью приватного ключа и RSA;

2) Удаление зашифрованного файла с ключом;

3) Считывание ключа из файла;

4) Удаление файла с ключом;

5) Расшифровка зашифрованной базы данных с помощью ключа и алгоритма AES;

6) Удаление зашифрованной базы данных.

Листинг 1.13.7

|  |  |
| --- | --- |
| 1  -  2 | string command = "openssl\\bin\\openssl.exe rsautl -decrypt -inkey rsa.private -in key.txt.enc -out key.txt";  system(command.c\_str()); |

Листинг 1.13.8

|  |  |
| --- | --- |
| 1  2  3 | if (remove("key.txt.enc") != 0) {  cout << "[ERROR] - deleting file" << endl;  } |

Листинг 1.13.9

|  |  |
| --- | --- |
| 1  2  3  4  5 | char\* pass = new char[64];  ifstream file;  file.open("key.txt", ios::binary);  file.read(pass, 65);  file.close(); |

Листинг 1.13.10

|  |  |
| --- | --- |
| 1  2  3 | if (remove("key.txt") != 0) {  cout << "[ERROR] - deleting file" << endl;  } |

Листинг 1.13.11

|  |  |
| --- | --- |
| 1  -  2  3 | command = "openssl\\bin\\openssl.exe enc -aes-256-cbc -d -in file.txt.enc -out file.txt -pass pass:";  command += pass;  system(command.c\_str()); |

Листинг 1.13.12

|  |  |
| --- | --- |
| 1  2  3 | if (remove("file.txt.enc") != 0) {  cout << "[ERROR] - deleting file" << endl;  } |

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Структура программы

Исходными данными для программы является информация о группе студентов из N человек, где запись о студенте содержит следующие данные (таблица 2.1.1).

Таблица 2.1.1 – Структура и типы данных

|  |  |
| --- | --- |
| Основная структура данных | Типы данных |
| Имя | std::string(строковый) |
| Отчество | std::string(строковый) |
| Фамилия | std::string(строковый) |
| Пол студента | std::string(строковый) |
| Число, месяц, год рождения | std::string(строковый) |
| Год поступления в институт | unsigned short(целочисленный) |
| Институт | std::string(строковый) |
| Кафедра | std::string(строковый) |
| Группа | std::string(строковый) |
| Номер зачетной книжки | std::string(строковый) |
| Название предмета | std::string(строковый) |
| Оценка | unsigned short(целочисленный) |

По условиям курсовой работы, допустимо максимально 9 сессий и 10 предметов в каждом семестре, которые могут быть разными. Все данные являются форматными.

Для реализации соответствующей задачи были разработаны классы. В данных классах описаны методы решения поставленной задачи.

В представленной ниже таблице продемонстрированы роли используемых классов (таблица 2.1.2).

Таблица 2.1.2 – Классы

|  |  |
| --- | --- |
| Класс | Назначение |
| List | Шаблонный линейный односвязный список. |
| Node | Структура узла линейного списка. |
| Student | Хранит персональные данные студента |
| grade | Содержит в себе информацию об оценке, полученной студентом |
| Exercise79 | Класс-задание курсовой работы, который выполняет требуемые действия. |

Для разработки данного проекта понадобятся следующие классы. В таблице ниже представлены и описаны поля и методы класса, используемые в проекте (таблица 2.1.3).

Таблица 2.1.3 – Поля классов

|  |  |  |
| --- | --- | --- |
| Классы | Поле класса | |
| Тип данных | Название и характеристика |
| List | Node\* first  Node\* last | Указатели на первый и последний элемент |
| Node | T unit | Информация в узле с произвольным типом данных |
| Unsigned int index | Индекс узла |
| Node\* next | Указатель на следующий узел |
| List | Node\* first | Указатель на первый узел списка |
| Node\* last | Указатель на последний узел списка |
| grade | std::string subject | Название дисциплины |
| dtd::string date | Дата оценки |
| Unsigned short mark | Сама оценка |
| Student | std::string number | Номер студента |
| Std::string first\_name | Имя студента |
| Std::string middle\_name | Отчество студента |
| Std::string last\_name | Фамилия студента |
| Unsigned short year | Год поступления студента |
| Std::string birthday | День рождения студента |
| Std::string faculty | Факультет |
| Std::string department | Кафедра |
| Std::string group | Группа |
| Std::string gender | Пол студента |
| List <grade> grades | Оценки студента |
| Exercise79 | List <student> database | Линейный односвязный список со студентами |

Далее методы классов, представленные в таблице 2.1.4.

Таблица 2.1.4 Методы классов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы класса | | | |
| Название | Назначение | Аргументы  (их тип) | Тип возвращаемого значения |
| List | Append | Добавляет в конец списка элемент | Const T newData | - |
| Remove | Удаляет элемент списка по индексу | Const unsigned int index | - |
| Find | Ищет элемент в списке и возвращает его индекс | T data | Unsigned int |
| Length | Возвращает длину списка | - | Unsigned int |
| isExists | Проверяет, существует ли элемент в списке | T data | Bool |

Продолжение таблицы 2.1.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы класса | | | |
| Название | Назначение | Аргументы  (их тип) | Тип возвращаемого значения |
| Exercise79 | toFile | Записывает базу данных в файл | const std::string filename | - |
| Append | Добавляет элемент в базу данных | Const Student unit | - |
| Remove | Удаляет элемент из базы данных по индексу | Const unsigned int index | - |
| Find | Ищет студента в базе данных и возвращает его индекс | const Student unit | Unsigned int |
| isExists | Говорит, занесен ли данный студент в базу данных | const Student unit | Bool |
| Split | Делит группу студентов на две части - тех, кто поступил в одном году, и тех, кто в другие | Std::string group | Std::pair <List <Student>, List<Student>> |
| get | Возвращает студента по индексу | Unsigned int index- | Student |
| sortGroup | Сортирует студентов по успеваемости | std::string group | List <Student> |
| sortList | Сортирует список с помощью Bubble Sort | List <Student> | - |
| Encrypt | Шифрует базу данных с помощью AES128 | std::string filenameIn, std::string filenameOut | - |
| Decrypt | Расшифровывает базу данных | std::string filenameIn, std::string filenameOut, | - |

2.2 Алгоритм решения задачи

Отсортировать всю группу по увеличению года

поступления в ВУЗ, с указанием интервала года рождения.

Данное задание было реализовано с помощью линейного списка, который сортируется с помощью Bubble Sort.

**sortGroup(std::string group)**:

* + Этот метод принимает строку group в качестве аргумента.
  + Создается пустой список result, который будет содержать объекты типа Student.
  + Затем происходит итерация по всей базе данных (database), хранящей студентов.
  + На каждой итерации проверяется, принадлежит ли группа студента (полученная через вызов метода getGroup()) указанной в аргументе group.
  + Если группа совпадает, студент добавляется в список result.
  + После завершения итерации по базе данных вызывается метод sortList(), чтобы отсортировать список result по году обучения студентов.
  + Наконец, отсортированный список result возвращается.

1. **sortList(List<Student>& unit) const**:
   * Этот метод принимает список студентов в качестве аргумента.
   * Он использует алгоритм сортировки пузырьком для сортировки списка студентов по году обучения (получаемому через вызов метода getYear()).
   * Вложенные циклы проходятся по списку и сравнивают каждую пару соседних элементов.
   * Если порядок элементов неправильный (элемент на более высоком курсе находится перед элементом на более низком курсе), они меняются местами.
   * Этот процесс продолжается до тех пор, пока список не будет полностью отсортирован по возрастанию года обучения.

2.3 Программная реализация задания

2.3.1 Пример выполнения задания

List <Student> Exercise79::sortGroup (std::string group) {

List <Student> result;

for (int i = 0; i < this->database.length(); ++i) {

if (this->database[i].getGroup() == group) {

result.append(this->database[i]);

}

}

this->sortList(result);

return result;

}

void Exercise79::sortList (List <Student>& unit) const {

for (int i = 0; i < unit.length() - 1; ++i) {

for (int j = 0; j < unit.length() - i - 1; ++j) {

if (unit[j].getYear() > unit[j + 1].getYear()) {

Student temp = unit[j];

unit[j] = unit[j + 1];

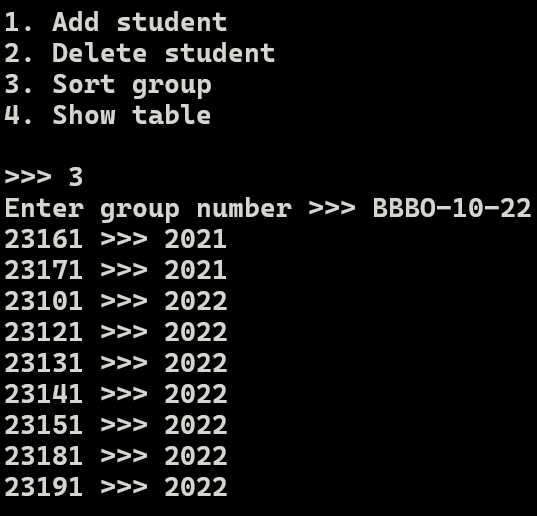
unit[j + 1] = temp;

}

}

}

}



2.3.2 Отображение списка студентов

Для отображения студентов используется operator<< в классе tolkachev

std::ostream& operator<< (std::ostream& os, tolkachev& unit) {

std::string toOut;

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

List <grade> grades = unit.database[i].getGrades();

for (int b = 0; b < grades.length(); ++b) {

std::string row =

unit.database[i].getNumber() + " " +

unit.database[i].getFirstName() + " " +

unit.database[i].getMiddleName() + " " +

unit.database[i].getLastName() + " " +

std::to\_string(unit.database[i].getYear()) + " " +

unit.database[i].getBirthday() + " " +

unit.database[i].getFaculty() + " " +

unit.database[i].getDepartment() + " " +

unit.database[i].getGroup() + " " +

unit.database[i].getGender() + " " +

grades[b].subject + " " +

grades[b].date + " " +

std::to\_string(grades[b].mark) + "\n";

toOut += row;

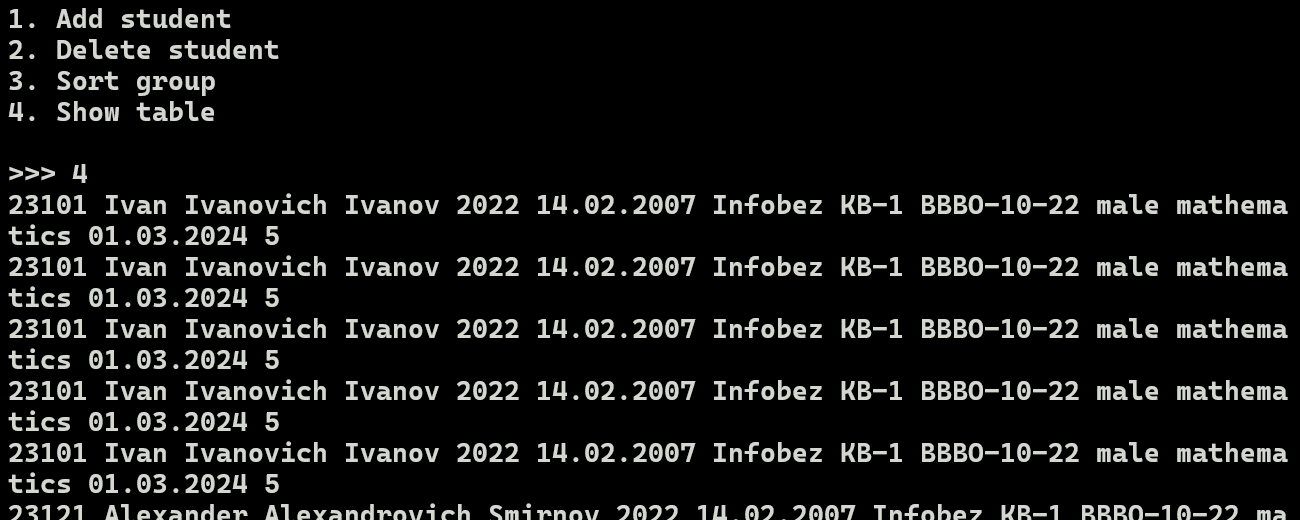
}

}

os << toOut;

return os;

}



2.3.3 Изменение данных о студенте

Данные о студенте изменяются по принципу замещения старого объекта студента в базе данных новым. Для этого сначала находится искомый нами студент с помощью метода find, после этого, сохранив его копию, мы выполняем изменения в этой копии, после изменений, с помощью метода append мы удаляем студента из базы данных, а после удаляем с помощью remove.

void tolkachev::append (const Student unit) {

this->database.append(unit);

}

void tolkachev::remove (const unsigned int index) {

this->database.remove(index);

}

unsigned int tolkachev::find (const Student unit) {

return this->database.find(unit);

}

bool tolkachev::isExists (const Student unit) {

return this->database.isExists(unit);

}

Student tolkachev::get (const unsigned int index) {

return this->database[index];

}

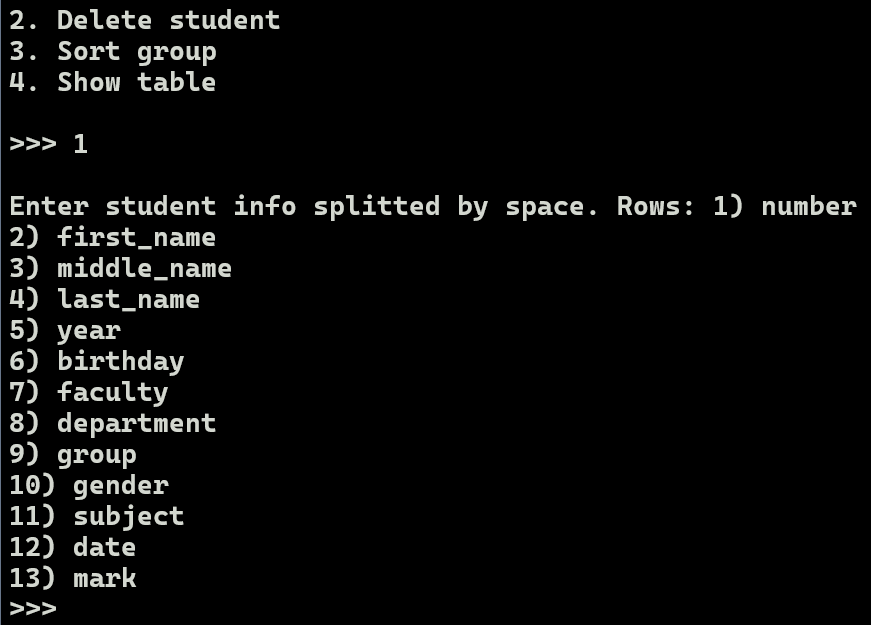
2.3.3 Добавление информации о новом студенте

Студент в базу данных добавляется с помощью метода append: создается объект класса Student, а после он передается в этот метод в качестве аргумента.

void tolkachev::append (const Student unit) {

this->database.append(unit);

}



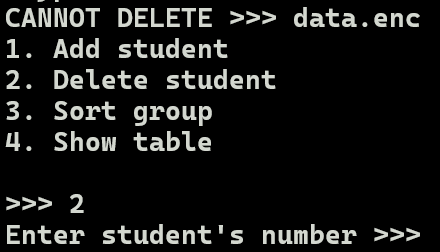
2.3.4 Удаление данных о студенте

Удаление данных о студенте происходит с помощью метода remove.

void Cursework::remove (const unsigned int index) {

this->database.remove(index);

}



2.3.5 Шифрование данных

Шифрование данных производится с помощью RSA и AES128. Генерируется случайный ключ, с помощью него шифруется файл алгоритмом AES128, а потом шифруется и сам ключ с помощью RSA.

void Exercise79::encrypt (const std::string filenameIn, const std::string filenameOut) {

std::string key;

char alphabet[63] = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz1234567890";

srand(time(NULL));

for (int i = 0; i < 16; ++i) {

key += alphabet[rand() % 62];

}

std::ofstream file("key.txt", std::ios::trunc);

file << key;

file.close();

std::string commandAES = "openssl aes-128-cbc -salt -in " + filenameIn + " -out " + filenameOut + " -pass pass:" + key;

std::system(commandAES.c\_str());

if (std::remove(filenameIn.c\_str()) != 0) {

std::cerr << "CANNOT DELETE >>> " + filenameIn << std::endl;

}

std::string commandRSA = "openssl rsautl -encrypt -inkey rsa.public -pubin -in key.txt -out key.txt.enc";

std::system(commandRSA.c\_str());

if (std::remove("key.txt") != 0) {

std::cerr << "CANNOT DELETE >>> key.txt" << std::endl;

}

}

void Exercise79::decrypt (std::string filenameIn, std::string filenameOut) {

std::string commandRSA = "openssl rsautl -decrypt -inkey rsa.private -in key.txt.enc -out key.txt";

std::system(commandRSA.c\_str());

std::string key;

std::ifstream file("key.txt", std::ios::in);

file >> key;

file.close();

if (std::remove("key.txt.enc") != 0) {

std::cerr << "CANNOT DELETE >>> key.txt.enc" << std::endl;

}

std::string commandAES = "openssl aes-128-cbc -d -in " + filenameIn + " -out " + filenameOut + " -pass pass:" + key;

std::system(commandAES.c\_str());

if (std::remove(filenameIn.c\_str()) != 0) {

std::cerr << "CANNOT DELETE >>> " + filenameIn << std::endl;

}

}

2.4 Руководство пользователя

При запуске программы появляется меню, в котором 6 пунктов

1. Add Student - Добавляет студента в базу данных
2. Remove Student - Удаляет студента из базы данных
3. Split group - Выполняет задание и разделяет группу на 2
4. Sort Parts of Groups – Выполняет задание и сортирует две части по успеваемости
5. Show table - Выводит всю базу данных на экран

2.5 Системные требования

Язык программирования: С++.

Операционная система: Linux Mint 20.1 выше или аналог, Windows 10.

ОЗУ: 1 Гб и более.

Свободное место на диске: 200 Мб и более.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы была написана программа, Разбить группу на 2 части:

1) студентов, поступивших в ВУЗ в одном и том же году;

2) студентов, поступивших в ВУЗ в др. годы, отличные от части

1.

Отсортировать каждую часть по успеваемости за все время обучения.

Кроме того, в процессе реализации программы мною были закреплены базовые навыки программирования, полученные при изучении дисциплины Языки программирования и языка программирования C++.

Были подробно рассмотрены теоретические выкладки, использованные в процессе написания программы.

Программа была реализована c использованием технологии ООП. Более того, была внедрена работа с динамической памятью, работа с файлами. Более того, детально были разобраны и применены на практике методы симметричного и ассиметричного шифрования и дешифрования файлов.

Дополнительно были созданы необходимые и достаточные условия для корректного выполнения индивидуального варианта задания.

По результату выполнения тестирования программного продукта, можно сделать вывод о том, что программа работает корректно и справляется с поставленной задачей.

Перспективы разработки данного продукта характеризуются высокой потребностью в универсальном и безопасном программном продукте, предназначенном для внедрения в средства автоматизации и информационные системы.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Мерсов А. А. Языки программирования [Электронный ресурс]: методические рекомендации по выполнению курсовой работы / А. А. Мерсов, А. М. Русаков, В. В. Филатов. — М.: РТУ МИРЭА, 2022. — Электрон. опт. диск (ISO)
2. Мерсов А. А. Основы объектно-ориентированного программирования на языке С++ [Электронный ресурс]: практикум / А. А. Мерсов, А. М. Русаков, В. В. Филатов. — М.: РТУ МИРЭА, 2021. — Электрон. опт. диск (ISO)
3. Страуструп Б. Программирование. Принципы и практика использования C++. – Litres, 2022.
4. Лафоре Р. Объектно-ориентированное программирование в C++. 4-е изд., перераб. и доп //Санкт-Петербург.: Питер. – 2022.
5. Стенли Липпман Язык программирования С++: полное руководство / Липпман Стенли, ЛажойеЖози. – Саратов: Профобразование, 2023. 1104 c.
6. Страуструп Б. Дизайн и эволюция языка С++. – Litres, 2022.
7. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. – Litres, 2022.
8. Аммерааль Л. STL для программистов на C++. – Litres, 2022.
9. Дейл Н., Уимз Ч., Хедингтон М. Программирование на C++. – Litres, 2022.
10. Нефедов Д. Г., Русяк И. Г., Вавилова Д. Д. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ. – 2020.
11. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – " Издательский дом"" Питер""", 2018.
12. Гудлиф П. Ремесло программиста. – 2009.

ПРИЛОЖЕНИЕ А Исходный код программы

Main.cpp

#include <iostream>

#include "List.h"

#include "Exercise79.h"

#include "Student.h"

void addStudent (Exercise79& unit) {

std::string studentInfo;

std::cout << std::endl << "Enter student info splitted by space. Rows: 1) number" << std::endl <<

"2) first\_name" << std::endl << "3) middle\_name" << std::endl << "4) last\_name" << std::endl <<

"5) year" << std::endl << "6) birthday" << std::endl << "7) faculty" << std::endl <<

"8) department" << std::endl << "9) group" << std::endl << "10) gender" << std::endl <<

"11) subject" << std::endl << "12) date" << std::endl << "13) mark" << std::endl << ">>> ";

std::cin >> studentInfo;

std::istringstream iss(studentInfo);

std::string token;

unsigned int count = 0;

Student student;

grade g;

while (std::getline(iss, token, ' ')) {

if (count == 0) { student.setNumber(token); }

else if (count == 1) { student.setFirstName(token); }

else if (count == 2) { student.setMiddleName(token); }

else if (count == 3) { student.setLastName(token); }

else if (count == 4) { student.setYear(std::stoi(token)); }

else if (count == 5) { student.setBirthday(token); }

else if (count == 6) { student.setFaculty(token); }

else if (count == 7) { student.setDepartment(token); }

else if (count == 8) { student.setGroup(token); }

else if (count == 9) { student.setNumber(token); }

else if (count == 10) { g.subject = token; }

else if (count == 11) { g.date = token; }

else if (count == 12) { g.mark = std::stoi(token); }

++count;

}

student.appendGrade(g);

unit.append(student);

std::cout << "Success!" << std::endl;

}

void deleteStudent (Exercise79& unit) {

std::string number;

std::cout << "Enter student's number >>> ";

std::cin >> number;

Student student;

student.setNumber(number);

if (unit.isExists(student)) {

unit.remove(unit.find(student));

std::cout << "Success!" << std::endl;

}

else {

std::cerr << "Student does not exists!" << std::endl;

}

}

void sortGroup (Exercise79& unit) {

std::string group;

std::cout << "Enter group number >>> ";

std::cin >> group;

List <Student> result = unit.sortGroup(group);

if (result.length() == 0) {

throw std::logic\_error("Group doesn't exist!");

}

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

std::cout << result[i].getNumber() << " >>> " << result[i].getYear() << std::endl;

}

}

int main () {

Exercise79::decrypt("data.enc", "data.txt");

Exercise79 unit("data.txt");

int choice;

std::cout <<

"1. Add student" << std::endl <<

"2. Delete student" << std::endl <<

"3. Sort group" << std::endl <<

"4. Show table" << std::endl << std::endl <<

">>> ";

std::cin >> choice;

if (std::cin.fail()) {

std::cerr << "You entered not num! Try again." << std::endl;

return 1;

}

switch (choice) {

case 1:

addStudent(unit);

break;

case 2:

deleteStudent(unit);

break;

case 3:

sortGroup(unit);

break;

case 4:

std::cout << unit << std::endl;

break;

default:

std::cerr << "You entered wrong num! Try again." << std::endl;

return 1;

}

unit.toFile("data.txt");

Exercise79::encrypt("data.txt", "data.enc");

return 0;

}

Student.h

#ifndef QWERTYUIOP

#define QWERTYUIOP

#include "List.h"

#include <string>

#include <ostream>

struct grade {

std::string subject;

std::string date;

unsigned short mark;

};

class Student {

private:

std::string number;

std::string first\_name;

std::string middle\_name;

std::string last\_name;

unsigned short year;

std::string birthday;

std::string faculty;

std::string department;

std::string group;

std::string gender;

List <grade> grades;

public:

Student () {};

Student (const Student& unit);

Student (

const std::string number, const std::string first\_name, const std::string middle\_name, const std::string last\_name,

const unsigned short year, const std::string birthday, const std::string faculty, const std::string department,

const std::string group, const std::string gender, const List <grade> grades

);

Student& operator= (const Student& unit);

bool operator== (const Student& unit1) const;

std::string getNumber () const { return this->number; }

std::string getFirstName () const { return this->first\_name; }

std::string getMiddleName () const { return this->middle\_name; }

std::string getLastName () const { return this->last\_name; }

unsigned short getYear () const { return this->year; }

std::string getBirthday () const { return this->birthday; }

std::string getFaculty () const { return this->faculty; }

std::string getDepartment () const { return this->department; }

std::string getGroup () const { return this->group; }

std::string getGender () const { return this->gender; }

List <grade> getGrades () const { return this->grades; }

void setNumber (const std::string data) { this->number = data; }

void setFirstName (const std::string data) { this->first\_name = data; }

void setMiddleName (const std::string data) { this->middle\_name = data; }

void setLastName (const std::string data) { this->last\_name = data; }

void setYear (const unsigned short data) { this->year = data; }

void setBirthday (const std::string data) { this->birthday = data; }

void setFaculty (const std::string data) { this->faculty = data; }

void setDepartment (const std::string data) { this->department = data; }

void setGroup (const std::string data) { this->group = data; }

void setGender (const std::string data) { this->gender = data; }

void appendGrade (grade data) { this->grades.append(data); }

};

#endif

Student.cpp

#include "Student.h"

Student::Student (const Student& unit) {

this->number = unit.number;

this->first\_name = unit.first\_name;

this->middle\_name = unit.middle\_name;

this->last\_name = unit.last\_name;

this->year = unit.year;

this->birthday = unit.birthday;

this->faculty = unit.faculty;

this->department = unit.department;

this->group = unit.group;

this->gender = unit.gender;

this->grades = unit.grades;

}

Student::Student (const std::string number, const std::string first\_name, const std::string middle\_name, const std::string last\_name, const unsigned short year, const std::string birthday, const std::string faculty, const std::string department, const std::string group, const std::string gender, const List <grade> grades) {

this->number = number;

this->first\_name = first\_name;

this->middle\_name = middle\_name;

this->last\_name = last\_name;

this->year = year;

this->birthday = birthday;

this->faculty = faculty;

this->department = department;

this->group = group;

this->gender = gender;

this->grades = grades;

}

Student& Student::operator= (const Student& unit) {

this->number = unit.number;

this->first\_name = unit.first\_name;

this->middle\_name = unit.middle\_name;

this->last\_name = unit.last\_name;

this->year = unit.year;

this->birthday = unit.birthday;

this->faculty = unit.faculty;

this->department = unit.department;

this->group = unit.group;

this->gender = unit.gender;

this->grades = unit.grades;

return \*this;

}

bool Student::operator== (const Student& unit1) const {

return (this->number == unit1.getNumber());

}

List.hpp

#ifndef LIST\_H

#define LIST\_H

#include <string>

#include <sstream>

#include <iostream>

#include <fstream>

template <typename T> struct Node {

T unit;

unsigned int index;

Node<T>\* next;

};

template <typename T> class List {

private:

Node<T>\* first;

Node<T>\* last;

public:

List ();

List (const List& unit);

List& operator= (const List& unit);

T& operator[] (const unsigned int index);

bool operator== (List& unit) const;

~List ();

void append (const T newData);

void remove (const unsigned int index);

unsigned int find (T data);

unsigned int length ();

bool isExists (T data);

};

template <typename T> List<T>::List () {

this->first = nullptr;

this->last = nullptr;

}

template <typename T> List<T>::List (const List<T>& unit) {

if (unit.first == nullptr) {

this->first = nullptr;

this->last = nullptr;

}

else {

Node<T>\* thisNode = new Node <T>;

this->first = thisNode;

Node<T>\* unitNode = unit.first;

while (true) {

thisNode->unit = unitNode->unit;

thisNode->index = unitNode->index;

if (unitNode->next == nullptr) {

thisNode->next = nullptr;

this->last = thisNode;

break;

}

else {

thisNode->next = new Node <T>;

unitNode = unitNode->next;

thisNode = thisNode->next;

}

}

}

}

template <typename T> List<T>& List<T>::operator= (const List<T>& unit) {

if (this != &unit) {

while (this->first != nullptr) {

Node<T>\* temp = this->first;

this->first = this->first->next;

delete temp;

}

if (unit.first == nullptr) {

this->first = nullptr;

this->last = nullptr;

}

else {

Node<T>\* thisNode = new Node <T>;

this->first = thisNode;

Node<T>\* unitNode = unit.first;

while (true) {

thisNode->unit = unitNode->unit;

thisNode->index = unitNode->index;

if (unitNode->next == nullptr) {

thisNode->next = nullptr;

this->last = thisNode;

break;

}

else {

thisNode->next = new Node <T>;

unitNode = unitNode->next;

thisNode = thisNode->next;

}

}

}

}

return \*this;

}

template <typename T> T& List<T>::operator[] (const unsigned int index) {

if (this->first == nullptr) {

throw std::logic\_error("List is empty");

}

else if (index < 0 || index > this->last->index) {

throw std::out\_of\_range("Index is out of range");

}

else {

Node<T>\* thisNode = this->first;

while (thisNode->index != index) {

thisNode = thisNode->next;

}

return thisNode->unit;

}

}

template <typename T> bool List<T>::operator== (List& unit) const {

return ((unit.first == this->first) and (unit.last == this->last));

}

template <typename T> List<T>::~List () {

if (this->first == nullptr) {

return;

}

Node<T>\* toDelete;

while (this->first->next != nullptr) {

toDelete = this->first;

this->first = this->first->next;

delete toDelete;

}

this->first = nullptr;

}

template <typename T> void List<T>::append (const T newData) {

Node<T>\* newNode = new Node <T>;

newNode->unit = newData;

newNode->next = nullptr;

if (this->first == nullptr) {

newNode->index = 0;

this->first = newNode;

}

else {

newNode->index = this->last->index + 1;

this->last->next = newNode;

}

this->last = newNode;

}

template <typename T> void List<T>::remove (const unsigned int index) {

if (this->first == nullptr) {

throw std::logic\_error("List is empty");

}

else if (index < 0 || index > this->last->index) {

throw std::out\_of\_range("Index is out of range");

}

else if (index == 0) {

if (this->first->next == nullptr) {

delete this->first;

this->first = nullptr;

this->last = nullptr;

}

else {

Node<T>\* thisNode = this->first->next;

delete this->first;

this->first = thisNode;

while (thisNode != nullptr) {

thisNode->index = thisNode->index - 1;

thisNode = thisNode->next;

}

}

}

else {

Node<T>\* thisNode = this->first;

if (this->last->index == index) {

while (thisNode->next->next != nullptr) {

thisNode = thisNode->next;

}

delete thisNode->next;

thisNode->next = nullptr;

this->last = thisNode;

}

else {

Node<T>\* lastNode = nullptr;

Node<T>\* toDelete = nullptr;

bool changeIndex = false;

while (thisNode->next != nullptr) {

if (changeIndex) {

thisNode->index = thisNode->index - 1;

thisNode = thisNode->next;

}

else {

if (thisNode->next->index == index) {

lastNode = thisNode;

toDelete = thisNode->next;

changeIndex = true;

}

thisNode = thisNode->next;

}

}

thisNode->index = thisNode->index - 1;

lastNode->next = toDelete->next;

delete toDelete;

}

}

}

template <typename T> unsigned int List<T>::find (T data) {

if (this->first == nullptr) {

throw std::logic\_error("List is empty");

}

else {

Node<T>\* thisNode = this->first;

while (thisNode != nullptr) {

if (thisNode->unit == data) {

return thisNode->index;

}

else {

thisNode = thisNode->next;

}

}

throw std::logic\_error("DATA HAS NOT FOUND");

}

}

template <typename T> unsigned int List<T>::length () {

if (this->last == nullptr) {

return 0;

}

else {

return this->last->index + 1;

}

}

template <typename T> bool List<T>::isExists (T data) {

if (this->first == nullptr) {

return false;

}

else {

Node<T>\* thisNode = this->first;

while (thisNode != nullptr) {

if (thisNode->unit == data) {

return true;

}

else {

thisNode = thisNode->next;

}

}

return false;

}

}

#endif

Exercise79.h

#ifndef EXERCISE\_79

#define EXERCISE\_79

#include "List.h"

#include <sstream>

#include <iostream>

#include <string>

#include <fstream>

#include <cstdio>

#include <map>

#include "Student.h"

#include <ctime>

class Exercise79 {

private:

List <Student> database;

public:

Exercise79 () {}

Exercise79 (const Exercise79& unit);

Exercise79 (const std::string filename);

Exercise79& operator= (const Exercise79& unit);

friend std::ostream& operator<< (std::ostream& os, Exercise79& unit);

~Exercise79 () {}

void toFile (const std::string filename);

void append (const Student unit);

void remove (const unsigned int index);

unsigned int find (const Student unit);

Student get (const unsigned int index);

bool isExists (const Student unit);

List <Student> sortGroup (std::string group);

void sortList (List <Student>& unit) const;

static void encrypt (const std::string filenameIn, const std::string filenameOut);

static void decrypt (const std::string filenameIn, const std::string filenameOut);

};

#endif // EXERCISE\_79

Exercise79.cpp

#include "Exercise79.h"

Exercise79::Exercise79 (const Exercise79& unit) {

this->database = unit.database;

}

Exercise79::Exercise79 (const std::string filename) {

std::ifstream file (filename, std::ios::in);

if (file.is\_open()) {

std::string row;

while (std::getline(file, row)) {

std::istringstream iss(row);

std::string word;

Student unit;

int count = 0;

grade g;

while (iss >> word) {

if (count == 0) { unit.setNumber(word); }

else if (count == 1) { unit.setFirstName(word); }

else if (count == 2) { unit.setMiddleName(word); }

else if (count == 3) { unit.setLastName(word); }

else if (count == 4) { unit.setYear(std::stoi(word)); }

else if (count == 5) { unit.setBirthday(word); }

else if (count == 6) { unit.setFaculty(word); }

else if (count == 7) { unit.setDepartment(word); }

else if (count == 8) { unit.setGroup(word); }

else if (count == 9) { unit.setGender(word); }

else if (count == 10) { g.subject = word; }

else if (count == 11) { g.date = word; }

else if (count == 12) { g.mark = std::stoi(word); }

++count;

}

if (this->database.isExists(unit)) {

this->database[this->database.find(unit)].appendGrade(g);

}

else {

unit.appendGrade(g);

this->database.append(unit);

}

}

}

else {

throw std::runtime\_error("File does not exist");

}

file.close();

}

Exercise79& Exercise79::operator= (const Exercise79& unit) {

this->database = unit.database;

return \*this;

}

std::ostream& operator<< (std::ostream& os, Exercise79& unit) {

std::string toOut;

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

List <grade> grades = unit.database[i].getGrades();

for (int b = 0; b < grades.length(); ++b) {

std::string row =

unit.database[i].getNumber() + " " +

unit.database[i].getFirstName() + " " +

unit.database[i].getMiddleName() + " " +

unit.database[i].getLastName() + " " +

std::to\_string(unit.database[i].getYear()) + " " +

unit.database[i].getBirthday() + " " +

unit.database[i].getFaculty() + " " +

unit.database[i].getDepartment() + " " +

unit.database[i].getGroup() + " " +

unit.database[i].getGender() + " " +

grades[b].subject + " " +

grades[b].date + " " +

std::to\_string(grades[b].mark) + "\n";

toOut += row;

}

}

os << toOut;

return os;

}

void Exercise79::toFile (const std::string filename) {

std::string toOut;

for (int i = 0; i < this->database.length(); ++i) {

List <grade> grades = this->database[i].getGrades();

for (int j = 0; j < grades.length(); ++j) {

std::string row =

this->database[i].getNumber() + " " +

this->database[i].getFirstName() + " " +

this->database[i].getMiddleName() + " " +

this->database[i].getLastName() + " " +

std::to\_string(this->database[i].getYear()) + " " +

this->database[i].getBirthday() + " " +

this->database[i].getFaculty() + " " +

this->database[i].getDepartment() + " " +

this->database[i].getGroup() + " " +

this->database[i].getGender() + " " +

grades[j].subject + " " +

grades[j].date + " " +

std::to\_string(grades[j].mark) + "\n";

toOut += row;

}

}

std::ofstream file(filename.c\_str(), std::ios::trunc);

file << toOut;

file.close();

}

void Exercise79::append (const Student unit) {

this->database.append(unit);

}

void Exercise79::remove (const unsigned int index) {

this->database.remove(index);

}

unsigned int Exercise79::find (const Student unit) {

return this->database.find(unit);

}

bool Exercise79::isExists (const Student unit) {

return this->database.isExists(unit);

}

Student Exercise79::get (const unsigned int index) {

return this->database[index];

}

List <Student> Exercise79::sortGroup (std::string group) {

List <Student> result;

for (int i = 0; i < this->database.length(); ++i) {

if (this->database[i].getGroup() == group) {

result.append(this->database[i]);

}

}

this->sortList(result);

return result;

}

void Exercise79::sortList (List <Student>& unit) const {

for (int i = 0; i < unit.length() - 1; ++i) {

for (int j = 0; j < unit.length() - i - 1; ++j) {

if (unit[j].getYear() > unit[j + 1].getYear()) {

Student temp = unit[j];

unit[j] = unit[j + 1];

unit[j + 1] = temp;

}

}

}

}

void Exercise79::encrypt (const std::string filenameIn, const std::string filenameOut) {

std::string key;

char alphabet[63] = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz1234567890";

srand(time(NULL));

for (int i = 0; i < 16; ++i) {

key += alphabet[rand() % 62];

}

std::ofstream file("key.txt", std::ios::trunc);

file << key;

file.close();

std::string commandAES = "openssl aes-128-cbc -salt -in " + filenameIn + " -out " + filenameOut + " -pass pass:" + key;

std::system(commandAES.c\_str());

if (std::remove(filenameIn.c\_str()) != 0) {

std::cerr << "CANNOT DELETE >>> " + filenameIn << std::endl;

}

std::string commandRSA = "openssl rsautl -encrypt -inkey rsa.public -pubin -in key.txt -out key.txt.enc";

std::system(commandRSA.c\_str());

if (std::remove("key.txt") != 0) {

std::cerr << "CANNOT DELETE >>> key.txt" << std::endl;

}

}

void Exercise79::decrypt (std::string filenameIn, std::string filenameOut) {

std::string commandRSA = "openssl rsautl -decrypt -inkey rsa.private -in key.txt.enc -out key.txt";

std::system(commandRSA.c\_str());

std::string key;

std::ifstream file("key.txt", std::ios::in);

file >> key;

file.close();

if (std::remove("key.txt.enc") != 0) {

std::cerr << "CANNOT DELETE >>> key.txt.enc" << std::endl;

}

std::string commandAES = "openssl aes-128-cbc -d -in " + filenameIn + " -out " + filenameOut + " -pass pass:" + key;

std::system(commandAES.c\_str());

if (std::remove(filenameIn.c\_str()) != 0) {

std::cerr << "CANNOT DELETE >>> " + filenameIn << std::endl;

}

}