

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«Московский технологический университет»**  **МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения**

**(ИППО)**

**КУРСОВАЯ РАБОТА**

по дисциплине: *Объектно-ориентированное программирование*

по профилю: *Технология разработки программного обеспечения*

направления профессиональной подготовки: *09.03.04. Программная инженерия, бакалавриат*

Тема: **«Разработка программы для учета успеваемости студентов в группе», реализация классов «Студент» и «Староста»**

Студент (ф.и.о. полностью): Янгуразов Рашид Алиевич

Группа: ИКБО-01-17

Работа представлена к защите\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись и ф.и.о. студента)

Руководитель: Лобанов Александр Анатольевич, к.т.н, доцент

Работа допущена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись и ф.и.о. студента)

Оценка по итогам защиты: \_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

**РЕФЕРАТ**

Курсовая работа содержит 21 страниц, 5 иллюстраций, 1 таблицу, 1 исполняемый файл программы, 1 приложение с исходными текстами программы на языке C++, а также презентацию по защите курсовой работы. При написании работы использовалось 6 источников.

Тема работы – разработка программы для учета успеваемости студентов в группе

Объект исследования – программное обеспечение, написанное на языке C++ с использованием объектно-ориентированного подхода.

Цель работы – освоить на практике основные подходы объектно-ориентированного программирования для разработки программ.

Что из себя будет представлять данное приложение?

Основное назначение данного приложения следующее: благодаря ему пользователь сможет в режиме реального времени контролировать успеваемость студентов отдельно взятой группы. Дополнительно в приложении пользователь может вести список студентов, взявших академический отпуск.

Для реализации функционала приложения никаких новых подходов к реализации алгоритмов программы выдвинуто не будет. Вместо этого в приложении найдет применение контейнер из библиотеки STL. А ключевые объекты программы, на которых будет завязан весь смысл приложения, будут представлены как отдельные типы данных.

Речь, конечно же, об объектах «Студент» и «Староста».

Основной упор при разработке делается на удобство и ожидаемый результат от программы. С точки зрения пользователя данная программа имеет учебный характер, создание же коммерческой версии не планируется.

Ключевые слова: АЛГОРИТМЫ, С++, STL, КОНТЕЙНЕР, БИБЛИОТЕКА, КЛАСС, УНАСЛЕДОВАННЫЙ КЛАСС, ОБЪЕКТ, ООП

**СОДЕРЖАНИЕ**

Введение5

Актуальность выбранной темы6

Конкретизация задачи6

Границы применимости программы6

Масштабируемость6

1. Анализ предметной области7-8

2. Анализ функциональных требований9-13

3. Разработка структур данных и алгоритмов14-15

3.1. Разработка функциональных классов14

3.2. UML-диаграмма классов15

4. Реализация программы16-17

4.1. Описание работы программного средства 16

4.2. Пример работы программы 16-17

Рекомендации к использованию ПО18

Заключение19

Список используемых источников20

Приложение А. Листинг программы на языке C++21-36

Приложение Б. Глоссарий37

**ВВЕДЕНИЕ**

Сегодня мы живём в таком мире, в котором практически невозможно обойтись без использования компьютерных технологий. Этому есть простое объяснение: вследствие неуклонного роста объемов различного рода информации нам становится всё более важно и необходимо знать и уметь, как правильно хранить, обрабатывать и обновлять эти накопившиеся данные.

Такое развитие событие привело к необходимости создавать компьютерные системы, которые облегчают людям работать с этой информацией. Как правило, для каждого вида информации используется своё специализированное программное обеспечение.

Такие системы используются повсеместно. В том числе и в высших учебных заведениях. Программе, которая точно должна быть полезна в учебных заведениях, и посвящена данная курсовая работа.

Ни для кого не секрет, что вести статистику успеваемости обучающихся – обязанность каждого заведения, предоставляющего образовательные услуги. Это влечет за собой хранение большого объема информации. Ее обработка на бумажных носителях – процесс возможный и реализуемый, но очень затратный в плане человеческих и материальных ресурсов.

**АКТУАЛЬНОСТЬ ВЫБРАННОЙ ТЕМЫ**

Безусловно, любую информацию можно хранить на бумажных носителях, однако когда объем информации возрастает, а структура усложняется, данный тип носителя резко теряет свои преимущества – гораздо выгоднее использовать компьютерные технологии с их неограниченными возможностями оперирования информацией. Поэтому при контролировании учебного процесса использование возможностей информационных технологий представляется самым оптимальным вариантом облегчить себе задачу.

**КОНКРЕТИЗАЦИЯ ЗАДАЧИ**

Необходимо разработать такую программу, которая позволит хранить данные о студентах и их успеваемости, а также редактировать и удалять информацию, переносить студентов в список тех, кто взял академический отпуск. В разработанном продукте также должны быть реализованы такие дополнительные функции, как сортировка студентов по рейтингу, отображение рейтинга отдельного конкретного студента, поиск студента, отображение контактных данных старосты.

**ГРАНИЦА ПРИМЕНИМОСТИ ПРОГРАММЫ**

Данное программное обеспечение может использоваться как сотрудниками ВУЗа, так и тем, кто не является сотрудником ВУЗа. В последнем случае таковым может оказаться староста группы.

**МАСШТАБИРУЕМОСТЬ**

Коммерческое развертывание данной программы не предполагается.

1. **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

В данной курсовой работе рассматривается реализация программного средства, которое позволяет работать со списком студентов, причем как обучающихся в ВУЗе в настоящий момент, так и тех, кто находится в академический отпуске. В отдельном месте также предполагается хранение информации о старосте. Необходимо спроектировать в программе такие функциональные части, которые вместе будут осуществлять хранение и обновление данных об этих студентах, а позволять определять, кто среди них является старостой, а кто находится в академическом отпуске.

В программе каждая запись обоих списков имеет следующие характеристики: фамилия, имя, отчество, возраст и средний балл в рейтинге. При этом, если студент перемещается в список тех, кто находится в академический отпуске, то у него рейтинг автоматически обнуляется. Для хранения информации о старосте в программе имеется отдельный тип записи с дополнительным полем «номер телефона».

При входе в программу данные можно будет считать как из файла, так и с клавиатуры. При каждом обновлении списка студентов, обучающихся в настоящий момент в ВУЗе, предоставляется возможность сохранять этот обновленный список в файл.

Для хранения всей обрабатываемой информации в программе будут использоваться два контейнера <vector>, который предоставляется библиотекой STL, и специальная переменная-объект, хранящая информацию только о старосте.

Для анализа предметной области также была представлена и протестирована аналогичная по функционалу программа стороннего разработчика. Она называется [**UUStud**](http://www.softportal.com/software-9694-uustud.html)**1.0.**

**UUStud 1.0** – программа, позволяющая вести учет студентов в высших учебных заведениях по результатам успеваемости по каждому отдельному предмету.

Его основные возможности следующие:

* Учет предметов
* Учет групп
* Учет студентов
* Учет оценок предметов
* Учет оценок экзаменов
* Вывод на печать каждого списка

Для оценки спроектированного ПО проведем сравнение с UUStud 1.0.

Таблица 1.Сравнение UUStud 1.0 с StudPro

|  |  |
| --- | --- |
| UUStud 1.0. | StudPRO |
| Отсутствие основного интерфейса | Присутствие основного интерфейса |
| Учет групп | Учет групп |
| Учет предметов | — |
| — | Рейтинг студента |
| — | Возраст студента |
| Учет оценок экзаменов | — |
| Учет оценок предметов | — |
| Учет студентов | Учет студентов |
| — | Быстрый поиск |
| — | Рейтинг определенного студента |
| — | Информация о старосте |
| — | Академический отпуск |
| Сохранение в файл | Сохранение в файл |
| — | Сортировка студентов по рейтингу |

1. **АНАЛИЗ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

Анализ функциональных требований будет предоставлен в виде Технического задания.

**Описание требований к разрабатываемой программной системе с точки зрения пользователя**

Необходимо разработать программу для ведения списка студентов актуальном состоянии. Каждая запись этого списка содержит сведения о конкретном студенте. Староста выделяется как отдельный тип записи.

1. **Введение**
   1. **Наименование программы:** "StudPro".
   2. **Назначение и область разработки:**

Программа предназначена для ведения списка студентов группы в актуальном состоянии

Программа доступна в консольной версии.

1. **Требования к программе**
   1. **Требования к функциональным возможностям:**

Программа должна обеспечивать возможность использования следующих функций:

* Считывание списка студентов с клавиатуры
* Считывание списка студентов с файла
* Редактирование имеющейся информации о конкретном студенте
* Удаление студента из списка в связи с отчислением
* Перемещение студента в список тех, кто взял академический отпуск
* Просмотр текущего списка студентов
* Сохранение текущего списка студентов в файл
* Определение старосты и ее контактных данных
* Просмотр контактных данных старосты
* Поиск студента в списке
* Рейтинга определенного студента
* Сортировка списка студентов по убыванию их рейтинга

Данные представляют собой информацию о фамилии, имени, отчестве студента, а также о его возрасте и среднем балле в рейтинге, а также номер телефона старосты.

Общий список требований к функциональным возможностям программного обеспечения представлен в методических указаниях преподавателей Московского технологического университета: Зориной Н.В., Зорина Л.Б., Соболева О.В [5].

* 1. **Требования к надежности**

**Требования к обеспечению надежного функционирования программы**

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

* организацией бесперебойного питания технических средств
* использованием лицензионного программного обеспечения
* регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»
* регулярным выполнением требований ГОСТ 51188-98.
* защитой информации.
* испытанием программных средств на наличие компьютерных вирусов
* соблюдением целостности программного кода (исходного вида)

1. **Условия эксплуатации**
   1. **Климатические условия эксплуатации**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

* 1. **Требования к составу и параметрам технических средств**

В состав технических средств должен входить IВМ-совместимый персональный компьютер (ПЭВМ), включающий в себя:

* процессор версии Intel Pentium, не менее;
* оперативную память объемом 1 Гигабайт, не менее;
* 40 Гигабайт на HDD, не менее;
* операционную систему Windows версии XP, не менее;
  1. **Требования к информационной и программной совместимости**

**Требования к исходным кодам и языкам программирования**

Программа написана на языке C++

**Требования к защите информации и программ**

Требования не предъявляются.

* 1. **Специальные требования**

Программа должна обеспечивать работу, а также сохранение в файл данных.

1. **Требования к программной документации**
   1. **Предварительный состав программной документации**

Состав программной документации должен включать в себя:

* техническое задание
* программу и методики испытаний
* руководство пользователя.

1. **Технико-экономические показатели**
   1. **Экономические преимущества разработки**

Ориентировочная экономическая эффективность не рассчитывается. Аналогия не проводится ввиду уникальности предъявляемых требований к разработке.

1. **Стадии и этапы разработки**
   1. **Стадии разработки**

Разработка должна быть проведена в три стадии:

* разработка технического задания
* рабочее проектирование
* внедрение
  1. **Этапы разработки**

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.   
На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

* разработка программы
* разработка программной документации
* испытания программы

На стадии внедрения должен быть выполнен этап разработки подготовка и передача программы.

* 1. **Содержание работ по этапам**

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

* постановка задачи
* определение и уточнение требований к техническим средствам
* определение требований к программе
* определение стадий, этапов и сроков разработки программы и документации на неё
* согласование и утверждение технического задания

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.  
На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

* разработка, согласование и утверждение и методики испытаний
* проведение приемо-сдаточных испытаний
* корректировка программы и программной документации по результатам испытаний

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах Заказчика.

1. **Порядок контроля и приемки**
   1. **Виды испытаний**

Приемо-сдаточные испытания должны проводиться на объекте Заказчика в оговоренные сроки.

Приемо-сдаточные испытания программы должны проводиться согласно разработанной Исполнителем и согласованной Заказчиком Программы и методик испытаний.

Ход проведения приемо-сдаточных испытаний Заказчик и Исполнитель документируют в Протоколе проведения испытаний.

* 1. **Общие требования к приемке работы**

На основании Протокола проведения испытаний Исполнитель совместно с Заказчиком подписывает акт приемки-сдачи программы в эксплуатацию.

1. **РАЗРАБОТКА СТРУКТУРЫ ДАННЫХ И АЛГОРИТМОВ**
2. **Разработка функциональных классов**

Данная программа содержит два класса. Класс <Student>. Является главным классом программы. Он содержит следующие поля: строковые surname – фамилия, name – имя, midname – отчество, целочисленные years – возраст, и с плавающей запятой points – рейтинг. У самого класса есть три конструктора. Первый – конструктор по-умолчанию, второй – конструктор копирования, а третий – конструктор с параметрами. Так же присутствуют get-методы, определенные для каждого поля. Нельзя было обойтись без метода, позволяющего сбросить поле points в ноль. Для удобства работы с объектами данного класса были перегружены некоторые стандартные операторы: ввода-вывода, присваивания. Перегрузка имеется и для оператора сравнения “больше”, но смысл этой перегрузки исключительно технический.

Также в программе присутствует класс-наследник <TeamLeader>. Он является производным от класса <Student>. <TeamLeader> наследует все поля, конструкторы и методы, за исключением одного - перегруженного оператора присваивания. Для <TeamLeader> он перегружен по-новому. У данного производного класса, помимо унаследованных элементов от <Student>, есть и свои собственные: поле строкового типа tel\_num – номер телефона; и некоторые методы, позволяющие оперировать непосредственно с объектами класса <TeamLeader>.

Для принятия ключевых решений был использован комплект специальной литературы по языку C++ [1-4].

Полная реализация программы представлена в *ПРИЛОЖЕНИИ А.*

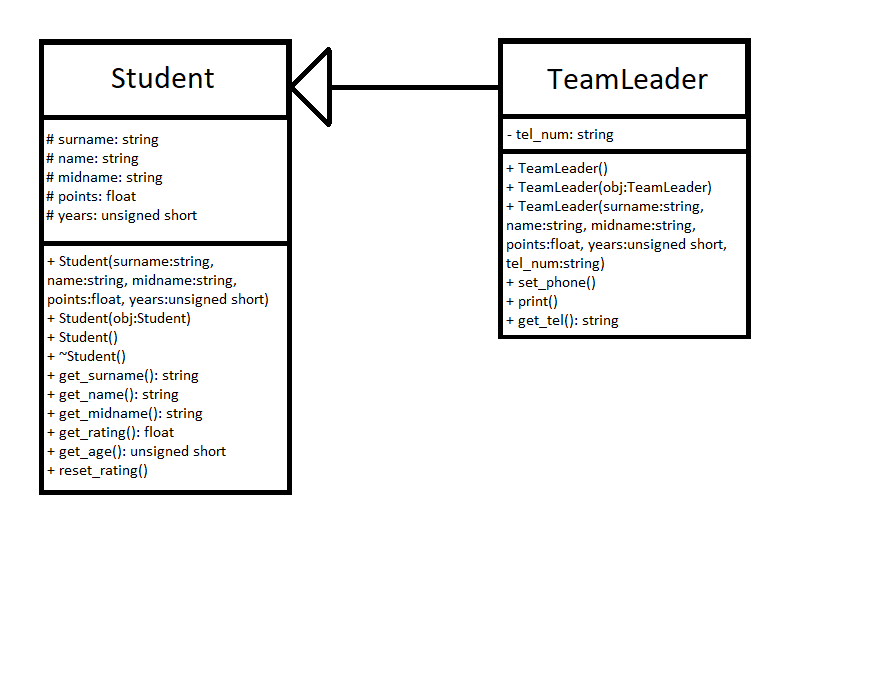
1. **UML-диаграмма классов**

Рисунок 3.1 UML-диаграмма отношения обобщения классов

На рисунке 3.1 представлена диаграмма обобщения двух классов. Обобщение (оно же наследование) выражает наследование, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя). Потомок разделяет структуру и поведение родителя. Графически обобщение представлено в виде сплошной линии с пустой стрелкой, указывающей на родителя [6].

1. **РЕАЛИЗАЦИЯ ПРОГРАММЫ**
2. **Описание работы программного средства**

При запуске программа сначала необходимо ввести в программу изначальный список студентов. При этом предлагается совершить импорт из файла, либо ввести напрямую с клавиатуры. И в том, и в другом случаях список заполняется в контейнер <vector>.

После того, как список оказывается в пределах досягаемости программы, она требует назначить старосту и уточнить ее контактный номер телефона. В это время для старосты выделяется отдельная переменная-объект класса <TeamLeader>.

После заполнения первоначальных данных программа переходит в основное меню. Здесь пользователю становятся доступными огромное количество действий из меню, которые можно совершить в этой программе. Меню будет появляться до тех пор, пока пользователь из него не выйдет (соответствующий пункт в меню имеется).

1. **Пример работы программы**

На рисунках 4.1, 4.2, 4.3, 4.4 ниже демонстрируется рабочее состояние программы, ее пользовательский интерфейс и пример вывода списка.



Рисунок 4.1 Окно первоначального заполнения контейнера <vector> списком студентов

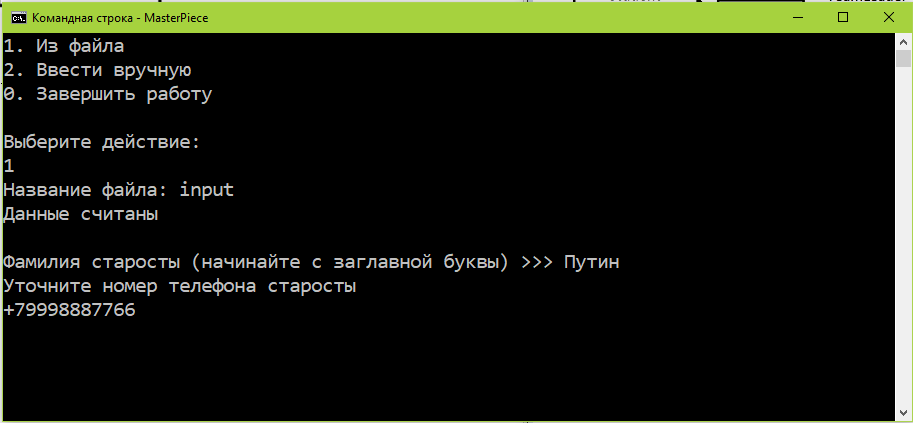


Рисунок 4.2 Назначение старосты

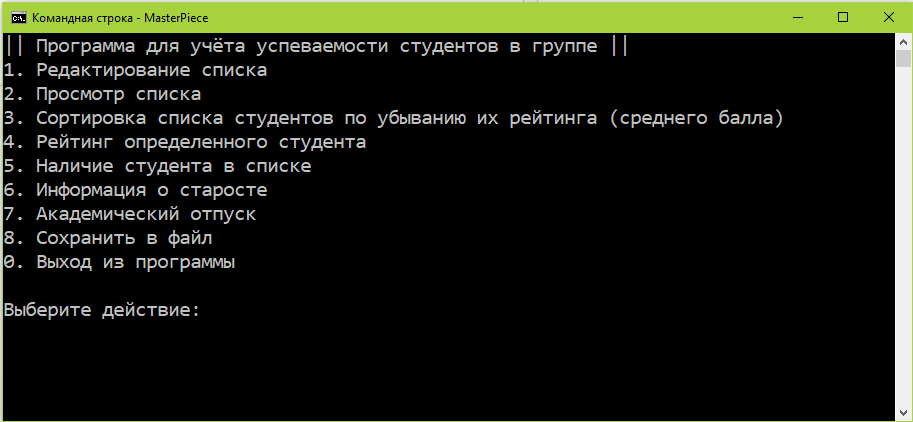


Рисунок 4.3 Главное меню программы

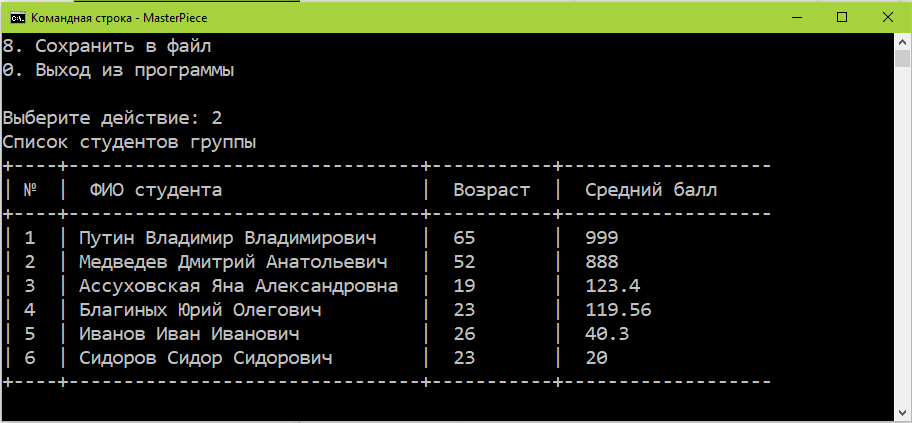


Рисунок.4.4 Отображение списка после сортировки в порядке убывания рейтинга

**РЕКОМЕНДАЦИИ И ИНСТРУКЦИИ ПО ИСПОЛЬЗОВАНИЮ ПО**

При использовании данного программного обеспечения необходимо и достаточно придерживаться следующих правил:

1. ФИО начинать заполнять строго с заглавной буквы.
2. В участках программы, где от пользователя требуется ввод числа, нужно вводить именно числовое значение.
3. Данные можно вводить как через пробел, так и через переход на новую строку.
4. Так как программа умеет работать только с текстовыми файлами одного единственного формата - txt, при уточнении имен этих файлов их расширение следует опустить.
5. Элементы списка в файле должно быть представлены согласно следующему шаблону:

<фамилия> <имя> <отчество> <рейтинг> <возраст>.

1. Каждый элемент в файле размещается c новой строки.
2. Староста должна быть в изначальном списке (как при использовании импортируемого файла, так и при вводе с клавиатуры).
3. Контактные данные старосты необходимо указывать сразу после первоначального ввода данных.
4. Текстовый файл, из которого планируется считывание данных, должен находиться в одном каталоге с программой.

**ЗАКЛЮЧЕНИЕ**

В ходе проделанной работы была создана полезная программа «StudPRO (Учет успеваемости студентов в группе)». Разработанная программа является приложением, позволяющим вести список студентов в актуальном состоянии, где каждая запись содержит сведения о конкретном студенте. Староста выделяется как отдельный тип записи. Программа имеет возможность добавления записи в коллекцию записей, редактирования имеющейся информации, а также для удаления записи. Она обеспечивает возможность реализации функции поиска, сортировки и определения рейтинга студентов в группе.

Программное средство содержит простой и интуитивно понятный интерфейс, пригодный для работы практически любого пользователя. Также приложение осуществляет контроль работы пользователя со стороны программы. В ходе разработки приложения было необходимо изучение дополнительной литературы с целью максимально правильной разработки программы.

Разработка приложения позволила систематизировать все знания, накопленные за пройденный курс, и обобщила уже имеющиеся. Разработка программного средства помогла разобраться в основных тонкостях программирования на языке С++, а также понять принципы объектно-ориентированного стиля.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Шилдт, Герберт. Полный справочник по С++ / Шилдт, Герберт.

– Пер. с англ. – М.: Вильямс, 2006 – 800 с. – ISBN 5-8459-0489-7.

1. Скляров В.А. Язык C++ и объектно-ориентированное программирование: справочное издание/ Скляров В.А. – Минск: Вышэйшая школа, 1997 – 478 с. – ISBN 985-06-0191-4.
2. Подбельский В.В. Язык C++: Учебное пособие / Подбельский В.В.

– 5-е изд. – М.: Финансы и статистика, 2000 – 560 с. – ISBN 5-279-02204-7.

1. Павловская Т.А. C/C++. Программирование на языке высокого уровня

/ Павловская Т.А. СПб.: Питер, 2006 – 461 с. – ISBN 5-94723-568-4.

1. Зорина Н.В., Зорин Л.Б., Соболев О.В. Методические указания по выполнению курсовой работы для бакалавров. Часть I: методические материалы для студентов дневного отделения / под ред. А.А. Хлебникова – М.: МИРЭА, 2017 – 42 с.
2. Вставская Е. UML-диаграммы классов / PROG-CPP.RU: Программирование на Си и С++ – 17.10.2017 – URL: http://prog-cpp.ru/uml-classes/ (дата обращения: 06.05.2018).

*ПРИЛОЖЕНИЕ А. Листинг программы на языке C++*

* Файл **main.cpp**

1. #include <iostream>
2. #include <vector>
3. #include <cstdlib>
4. #include <algorithm>
5. #include <functional>
6. #include "sysfunctions.h"
7. #include "newtypes.h"
9. **int** main() {
10. **float** average\_score = 0;
11. unsigned **char** item = 0;
12. **bool** flag = **false**, flag\_x = **false**;
13. std::vector <Student> list; //список студентов
14. std::vector <Student> academ\_vacation; //список студентов в академическом отпуске
15. //импорт списка или досрочный выход из программы
16. **do** {
17. item = menu\_of\_fill();
18. **switch** (item) {
19. **case** 48:
20. goodbye\_message();
21. flag = **false**;
22. **break**;
23. **case** 49:
24. flag = from\_the\_file(list); //чтение из файла
25. **break**;
26. **case** 50:
27. flag = keyboard(list);
28. **break**;
29. **default**:
30. try\_message();
31. **break**;
32. }
33. } **while** (item != 48 && !flag);
34. //определение старосты и вход в главное меню программы
35. **if** (flag) {
36. //определение старосты
37. flag = **false**;
38. TeamLeader starosta;
39. **do** {
40. flag = who\_is\_lead(list, starosta);
41. **if** (flag) {
42. std::cout << "Уточните номер телефона старосты" << std::endl;
43. starosta.set\_phone();
44. }
45. } **while** (!flag);
47. //вход в главное меню программы
48. **do**
49. {
50. item = user\_menu();
51. **switch** (item) {
52. **case** 48: //выход из программы
53. goodbye\_message();
54. **break**;
55. **case** 49:
56. **do** {
57. item = menu\_of\_editing();
58. **switch** (item) {
59. **case** 48:
60. **break**;
61. **case** 49:
62. std::cout << std::endl;
63. keyboard(list);
64. **break**;
65. **case** 50:
66. std::cout << std::endl;
67. flag\_x = **false**;
68. flag\_x = removing(list);
69. **if** (!flag\_x) {
70. std::cout << "В списке нет такого студента" << std::endl;
71. std::cin.get(); std::cin.get();
72. }
73. **break**;
74. **case** 51:
75. std::cout << std::endl;
76. flag\_x = **false**;
77. flag\_x = transfer(list, academ\_vacation);
78. **if** (!flag\_x) {
79. std::cout << "В списке нет такого студента" << std::endl;
80. std::cin.get(); std::cin.get();
81. }
82. **break**;
83. **case** 52:
84. std::cout << std::endl;
85. edit\_info(list);
86. **break**;
87. **default**:
88. try\_message();
89. **break**;
90. }
91. } **while** (item != 48);
92. item = 0;
93. **break**;
94. **case** 50:
95. std::cout << "Список студентов группы" << std::endl;
96. print\_students(list);
97. std::cin.get(); std::cin.get();
98. **break**;
99. **case** 51:
100. std::cout << "Отсортировано." << std::endl;
101. std::sort(list.begin(), list.end(), std::greater<Student>());
102. std::cin.get(); std::cin.get();
103. **break**;
104. **case** 52:
105. average\_score = define\_rating(list);
106. **if** (average\_score != 0) {
107. std:: cout << "Рейтинг студента равен " << average\_score << " баллам" << std::endl;
108. }
109. **else** {
110. std::cout << "В списке нет такого студента" << std::endl;
111. }
112. std::cin.get(); std::cin.get();
113. **break**;
114. **case** 53:
115. **if** (student\_search(list)) {
116. std:: cout << "В списке такой студент имеется" << std::endl;
117. }
118. **else** {
119. std:: cout << "В списке нет такого студента" << std::endl;
120. }
121. std::cin.get(); std::cin.get();
122. **break**;
123. **case** 54:
124. system("cls");
125. std::cout << "||Информация о старосте||" << std::endl;
126. starosta.print();
127. std::cin.get(); std::cin.get();
128. **break**;
129. **case** 55:
130. **if** (academ\_vacation.size() > 0) {
131. std::cout << "Студенты, взявшие академический отпуск"<< std::endl;
132. print\_students(academ\_vacation);
133. }
134. **else** {
135. std::cout << "Таких пока нет" << std::endl;
136. }
137. std::cin.get(); std::cin.get();
138. **break**;
140. **case** 56:
141. save\_to\_file(list, starosta);
142. **break**;
144. **default**:
145. try\_message();
146. **break**;
147. }
148. } **while** (item != 48);
149. }
150. **return** 0;
151. }

* Файл **newtypes.h**

1. #ifndef NEWTYPES\_H
2. #define NEWTYPES\_H
3. #include <iostream>
4. **class** Student {
5. **protected**:
6. std::string surname;
7. std::string name;
8. std::string midname;
9. **float** points;
10. unsigned **short** years;
11. **public**:
12. //конструктор с параметрами
13. Student(**const** std::string &, **const** std::string &, **const** std::string &, **const** **float** &, **const** unsigned **short** &);
14. //конструктор копирования
15. Student(**const** Student &);
16. //конструктор по-умолчанию
17. Student();
18. //деструктор
19. ~Student();
21. //перегрузка операторов ввода-вывода
22. **friend** std:: istream& operator >> (std:: istream &, Student &);
23. **friend** std:: ostream& operator << (std:: ostream &, Student &);
25. //перегрузка оператора присваивания
26. Student& operator =(**const** Student &other);
28. **friend** **const** **bool** operator > (**const** Student &, **const** Student &);
30. //объявления методов класса
31. std::string get\_surname() **const**; //получить фамилию
32. std::string get\_name() **const**; //получить имя
33. std::string get\_midname() **const**; //получить отчество
35. **float** get\_rating() **const**; //получить рейтинг
36. unsigned **short** get\_age() **const**; //получить возраст
38. **void** reset\_rating(); //сброс рейтинга
40. };
42. **class** TeamLeader: **public** Student {
43. **private**:
44. std::string tel\_num;
45. **public**:
46. //конструктор по-умолчанию
47. TeamLeader();
48. //конструктор копирования
49. TeamLeader(**const** TeamLeader &);
50. //конструктор с параметрами
51. TeamLeader(**const** std::string &, **const** std::string &, **const** std::string &, **const** **float** &, **const** unsigned **short** &, **const** std::string &);
53. //перегрузки оператора присваивания
54. TeamLeader &operator =(**const** Student &);
55. TeamLeader &operator =(**const** TeamLeader &);
57. **void** set\_phone(); //установить номер телефона
58. **void** print(); //распечатать информацию о старосте
59. std::string get\_tel() **const**; //получить номер телефона старосты
60. };
62. #endif

* Файл **sysfunctions.h**

1. #ifndef SYSFUNCTIONS\_H
2. #define SYSFUNCTIONS\_H
3. #include <vector>
4. #include "newtypes.h"
6. **bool** keyboard(std::vector <Student> &); //считывание студентов в список
8. **bool** from\_the\_file(std:: vector <Student> &); //считывание списка студентов из файлов
10. unsigned **char** menu\_of\_fill(); //меню заполнения
12. unsigned **char** user\_menu(); //главное меню
14. unsigned **char** menu\_of\_editing(); //меню редактирования
16. std::string del\_rubbish(std::string &); //оставляет в строке только буквы
18. **bool** removing(std::vector <Student> &); //удаление;
20. **bool** transfer(std::vector <Student> &, std::vector <Student> &); //перемещение студента в список отпускников
22. **void** edit\_info(std::vector <Student> &); //редактирование информации о конкретном студенте
24. **float** define\_rating(**const** std::vector <Student> &);
26. **bool** who\_is\_lead(**const** std::vector <Student> &, TeamLeader &);
28. std::string control\_phone(std::string &); //оставляет в строке только цифры и знак "+"
30. **bool** student\_search(**const** std::vector <Student> &);
32. **void** goodbye\_message();
34. **void** try\_message();
36. **void** print\_students(std::vector <Student> &);
38. //функции для записи данных в файл
40. **void** save\_to\_file(**const** std::vector <Student> &, **const** TeamLeader &);
42. **int** max\_lgth(**const** std::vector <Student> &);
44. **int** lgth(**const** Student &);
45. #endif

* Файл **newtypes.cpp**

1. #include "newtypes.h"
2. #include <string>
3. #include <cstring>
4. #include <iostream>
5. #include <limits>
6. #include "sysfunctions.h"
7. //полный конструктор
8. Student::Student(**const** std::string &sur,
9. **const** std::string &nam,
10. **const** std::string &mid,
11. **const** **float** &rating,
12. **const** unsigned **short** &age) {
14. surname = sur;
15. name = nam;
16. midname = mid;
17. points = rating;
18. years = age;
19. }
21. //конструктор по-умолчанию
22. Student::Student() {
23. points = 0;
24. years = 0;
25. }
27. //конструктор копировани¤
28. Student::Student(**const** Student &T) {
29. surname = T.surname;
30. name = T.name;
31. midname = T.midname;
32. points = T.points;
33. years = T.years;
34. }
36. //деструктор
37. Student::~Student() {
38. }
40. //перегрузка операторов ввода-вывода
41. std::istream& operator >>(std::istream &in, Student &obj){
42. **const** **int** N = 3;
43. std::string \*temp[N] = {&obj.surname, &obj.name, &obj.midname};
44. **for** (**int** i = 0; i < N; i++) {
45. in >> \*temp[i];
46. \*temp[i] = del\_rubbish(\*temp[i]);
47. }
48. in >> obj.points >> obj.years;
49. **return** in;
50. }
52. std::ostream& operator <<(std::ostream &out, Student &obj){
53. out << obj.surname << " "
54. << obj.name << " "
55. << obj.midname << ", "
56. << obj.points << ", "
57. << obj.years;
58. **return** out;
59. }
61. //перегрузка оператора присваивани¤
62. Student& Student::operator =(**const** Student &other){
63. **if** (**this** != &other) {
64. surname = other.surname;
65. name = other.name;
66. midname = other.midname;
67. points = other.points;
68. years = other.years;
69. }
70. **return** \***this**;
71. }
73. //перегрузка оператора сравнени¤
74. **const** **bool** operator > (**const** Student &x, **const** Student &y) {
75. **return** x.points > y.points;
76. }
78. //определения методов класса
80. std::string Student::get\_surname() **const** {
81. **return** surname;
82. }
84. std::string Student::get\_name() **const** {
85. **return** name;
86. }
88. std::string Student::get\_midname() **const** {
89. **return** midname;
90. }
92. **float** Student::get\_rating() **const** {
93. **return** points;
94. }
96. unsigned **short** Student::get\_age() **const** {
97. **return** years;
98. }
100. **void** Student::reset\_rating() {
101. points = 0;
102. }
104. //конструктор по-умолчанию
105. TeamLeader::TeamLeader() : Student() {
106. }
107. //конструктор копировани¤
108. TeamLeader::TeamLeader(**const** TeamLeader &x) : Student(x) {
109. tel\_num = x.tel\_num;
110. }
111. //конструктор с параметрами
112. TeamLeader::TeamLeader(**const** std::string &sur, **const** std::string &nam,
113. **const** std::string &mid, **const** **float** &rating,
114. **const** unsigned **short** &age, **const** std::string &t\_n) :
115. Student(sur, nam, mid, rating, age) {
116. tel\_num = t\_n;
117. }
119. TeamLeader& TeamLeader::operator =(**const** Student &obj) {
120. Student::operator =(obj);
121. }
123. TeamLeader& TeamLeader::operator =(**const** TeamLeader &obj) {
124. Student::operator =(obj);
125. tel\_num = obj.tel\_num;
126. }
128. **void** TeamLeader::set\_phone() {
129. std::string str\_num;
130. std::cin >> str\_num;
131. str\_num = control\_phone(str\_num);
132. tel\_num = str\_num;
133. }
135. **void** TeamLeader::print() {
136. std::cout << "Фамилия: " << surname << std::endl;
137. std::cout << "Имя: " << name << std::endl;
138. std::cout << "Отчество: " << midname << std::endl;
139. std::cout << "Средний балл в рейтинге: " << points << std::endl;
140. std::cout << "Возраст: " << years << std::endl;
141. std::cout << "Тел.: " << tel\_num;
142. }
144. std::string TeamLeader::get\_tel() **const** {
145. **return** tel\_num;
146. }

* Файл **sysfunctions.cpp**

1. #include <iostream>
2. #include <vector>
3. #include <iomanip>
4. #include <cstdlib>
5. #include <cctype>
6. #include <fstream>
7. #include <sstream>
8. #include "sysfunctions.h"
9. #include "newtypes.h"
10. #include <string>
11. #include <clocale>
13. **static** **int** qty() {
14. std::cout << "Сколько человек вы хотите добавить в список? >>> ";
15. **int** n; std::cin >> n;
16. **return** n;
17. }
19. **bool** keyboard(std::vector <Student> &list) { //считывание списка студентов с клавиатуры
20. **int** n = qty();
21. **if** (n > 0) {
22. std:: cout << "Введите ФИО, рейтинг (может быть дробным) и возраст через пробел, затем нажмите Enter" << std::endl;
23. **for** (**int** i = 0; i < n; i++) {
24. Student temp;
25. std::cin >> temp;
26. list.push\_back(temp);
27. }
28. std::cout << "Конец" << std::endl;
29. std::cin.get();std::cin.get();
30. **return** **true**;
31. }
32. **return** **false**;
33. }
35. **bool** from\_the\_file(std::vector <Student> &list) { //считывание списка студентов из файла
37. std::cout << "Название файла: ";
38. std::string name\_of\_file;
39. std::cin >> name\_of\_file;
40. name\_of\_file += ".txt";
42. std::ifstream fin;
43. std::string sn, name, mn, str; **float** points\_; **int** years\_;
44. **int** n;
45. std::stringstream ss;
47. fin.open(name\_of\_file, std::ios::in);
48. **if** (!fin.is\_open()) { //если файл не открыт
49. std::cout << "Не удаётся открыть файл\n";
50. std::cin.get();std::cin.get();
51. **return** **false**;
52. }
53. **if** (fin.peek() == EOF) { //если файл не открыт
54. std::cout << "Файл пусть\n";
55. std::cin.get();std::cin.get();
56. **return** **false**;
57. }
58. **while** (getline(fin, str)) {
59. ss << str;
60. ss >> sn >> name >> mn >> points\_ >> years\_;
61. sn = del\_rubbish(sn);
62. name = del\_rubbish(name);
63. mn = del\_rubbish(mn);
64. Student temp(sn, name, mn, points\_, years\_);
65. list.push\_back(temp);
66. ss.clear();
67. }
68. std::cout << "Данные считаны" << std::endl;
69. std::cin.get();std::cin.get();
70. fin.close(); //закрываем файл
71. **return** **true**;
72. }
74. unsigned **char** menu\_of\_fill() { //меню заполнения
75. system("cls");
76. std:: string menu[] = {"Импорт списка", "1. Из файла", "2. Ввести вручную", "0. Завершить работу"};
77. **const** **int** size\_arr = **sizeof**(menu) / **sizeof**(std::string);
78. **for** (**int** i = 0; i < size\_arr; i++) {
79. std:: cout << menu[i] << std:: endl;
80. }
81. std:: cout << std:: endl << "Выберите действие: ";
82. unsigned **char** answer = 0;
83. std:: cin >> answer;
84. **return** answer;
85. }
87. unsigned **char** user\_menu(){ //главное меню
88. system("cls");
89. std:: string menu[] = {"|| Программа для учёта успеваемости студентов в группе ||",
90. "1. Редактирование списка", "2. Просмотр списка",
91. "3. Сортировка списка студентов по убыванию их рейтинга (среднего балла)",
92. "4. Рейтинг определенного студента", "5. Наличие студента в списке",
93. "6. Информация о старосте", "7. Академический отпуск", "8. Сохранить в файл",
94. "0. Выход из программы"};
95. **const** **int** size\_arr = **sizeof**(menu) / **sizeof**(std::string);
96. **for** (**int** i = 0; i < size\_arr; i++) {
97. std:: cout << menu[i] << std:: endl;
98. }
99. std:: cout << std:: endl << "Выберите действие: ";
100. **char** answer = 0;
101. std:: cin >> answer;
102. **return** answer;
103. }
105. unsigned **char** menu\_of\_editing() { //меню редактирования
106. system("cls");
107. std:: string menu[] = {"Редактирование списка", "1. Расширить список", "2. Удалить студента из списка в связи с отчислением",
108. "3. Переместить в список студентов в академическом отпуске",
109. "4. Отредактировать информацию о студенте", "0. Назад в главное меню"};
110. **const** **int** size\_arr = **sizeof**(menu) / **sizeof**(std::string);
111. **for** (**int** i = 0; i < size\_arr; i++) {
112. std:: cout << menu[i] << std:: endl;
113. }
114. std:: cout << std:: endl << "Выберите действие: ";
115. unsigned **char** answer = 0;
116. std:: cin >> answer;
117. **return** answer;
118. }
120. std:: string del\_rubbish(std::string &obj) { //оставляет в строке только буквы
121. setlocale(LC\_ALL, "rus");
122. std::string tmp = "\0";
123. **for** (**int** i = 0; i < obj.size(); i++) {
124. **if** ((i == 0) && (obj[i] == '-')) {
125. **break**;
126. }
127. **if** ((isalpha(obj[i])) || (obj[i] == '-')
128. || (obj[i]) == 'я' || (obj[i] == 'Я'))  {
129. tmp.push\_back(obj[i]);
130. }
131. }
132. **return** tmp;
133. }
135. **bool** removing(std::vector <Student> &list) { //отчисление студента
136. std::cout << "Фамилия студента, который отчислился (начинайте вводить с заглавной буквы) >>> ";
137. std::string sur;
138. std::cin >> sur;
139. sur = del\_rubbish(sur);
140. **for** (**int** i = 0; i < list.size(); i++) {
141. **if** (sur == list[i].get\_surname()) {
142. list.erase(list.begin() + i);
143. **return** **true**;
144. }
145. }
146. **return** **false**;
147. }
149. **bool** transfer(std::vector <Student> &list, std::vector <Student> &academ\_vacation) { //перемещение в список отпускников
150. std::cout << "Фамилия студента, взявшего академический отпуск (начинайте вводить с заглавной буквы) >>> ";
151. std::string sur;
152. std::cin >> sur;
153. sur = del\_rubbish(sur);
154. **for** (**int** i = 0; i < list.size(); i++) {
155. **if** (sur == list[i].get\_surname()) {
156. list[i].reset\_rating();
157. academ\_vacation.push\_back(list[i]);
158. list.erase(list.begin() + i);
159. **return** 1;
160. }
161. }
162. **return** 0;
163. }
165. **void** edit\_info(std::vector <Student> &list) { //редактирование информации о конкретном студенте
166. std::cout << "Студент, чьи данные требуется отредактировать, в списке имеет номер >>> ";
167. **int** n; std::cin >> n;
168. **if** (n-1 < list.size() && n > 0) {
169. std::cout << "Сейчас вы сможете полностью изменить информацию о студенте" << std::endl;
170. system("pause");
171. system("cls");
172. std:: cout << std::endl << "Введите ФИО, рейтинг (может быть дробным) и возраст через пробел, затем нажмите Enter" << std::endl;
173. Student temp; std::cin >> temp;
174. list[n-1] = temp;
175. }
176. **else** {
177. std::cout << "Максимально допустимый номер - это " << list.size() << "." << std::endl;
178. std::cin.get(); std::cin.get();
179. }
180. }
182. **float** define\_rating(**const** std::vector <Student> &list) { //функция, позволяющая узнать рейтинг студента по его фамилии
183. std::cout << "Фамилия студента, чей рейтинг нужно узнать (начинайте с заглавной буквы) >>> ";
184. std::string sur;
185. std::cin >> sur;
186. sur = del\_rubbish(sur);
187. **for** (**int** i = 0; i < list.size(); i++) {
188. **if** (sur == list[i].get\_surname()) {
189. **return** list[i].get\_rating();
190. }
191. }
192. **return** 0;
193. }
195. **bool** who\_is\_lead(**const** std::vector <Student> &list, TeamLeader &obj) { //определяет старосту
196. std::cout << "Фамилия старосты (начинайте с заглавной буквы) >>> ";
197. std::string sur; std::cin >> sur;
198. sur = del\_rubbish(sur);
199. **for** (**int** i = 0; i < list.size(); i++) {
200. **if** (sur == list[i].get\_surname()) {
201. obj = list[i];
202. **return** **true**;
203. }
204. }
205. **return** **false**;
206. }
208. std::string control\_phone(std::string &obj) { //удаляет из номера телефона лишние символы
209. std::string tmp = "\0";
210. **for** (**int** i = 0; i < obj.size(); i++) {
211. **if** ((i == 0) && (obj[i] == '-')) {
212. **continue**;
213. }
214. **else** **if** ((i == 0) && (obj[i] == '+')) {
215. tmp.push\_back(obj[i]);
216. }
218. **if** ((isdigit(obj[i])) || (obj[i] == '-' ))  {
219. tmp.push\_back(obj[i]);
220. }
221. }
222. **return** tmp;
223. }
225. **bool** student\_search(**const** std::vector <Student> &list) { //позволяет проверить наличие студента в списке
226. std::cout << "Фамилия студента, которого нужно найти (начинайте с заглавной буквы) >>> ";
227. std::string sur; std::cin >> sur;
228. sur = del\_rubbish(sur);
229. **for** (**int** i = 0; i < list.size(); i++) {
230. **if** (sur == list[i].get\_surname()) {
231. **return** **true**;
232. }
233. }
234. **return** **false**;
235. }
237. **void** goodbye\_message() { //прощальное сообщение перед выходом из программы
238. std::cout << "Работа программы завершена. Всего доброго" << std::endl;
239. }
241. **void** try\_message() {
242. std::cout << "Такого действия нет. Повторите попытку. " << std::endl;
243. system("pause");
244. system("cls");
245. }
247. **void** print\_students(std::vector <Student> &list){ //позволяет напечатать весь список студентов
248. **int** max = max\_lgth(list);
250. std::cout << "+----+";
251. **for** (**int** i = 0; i < max+2; i++) {
252. std::cout << "-";
253. }
254. std::cout << "+-----------+-------------------" << std::endl;
256. std::cout << "| №  | " << std::setw(max+1) << std::left << " ФИО студента";
257. std::cout << "|  Возраст  |  Средний балл  " << std::endl;
259. std::cout << "+----+";
260. **for** (**int** i = 0; i < max+2; i++) {
261. std::cout << "-";
262. }
263. std::cout << "+-----------+-------------------" << std::endl;
265. **for** (**int** i = 0; i < list.size(); i++) {
267. **if** (list.size() < 9) {
268. std::cout << "| " << i+1 << "  |";
269. }
270. **else** **if** ((list.size() >= 9) && (list.size() < 99)) {
271. std::cout << "| " << i+1 << " |";
272. }
273. std::cout << " " << list[i].get\_surname() << " " << list[i].get\_name() << " " << list[i].get\_midname() << " ";
274. **int** size = lgth(list[i]);
275. **for** (**int** i = 0; i < max - size+1;i++) {
276. std::cout << " ";
277. }
279. std::cout << "|  " << list[i].get\_age();
281. **if** (list[i].get\_age() < 10) {
282. **for** (**int** i = 0; i < 8; i++) {
283. std::cout << " ";
284. }
285. }
286. **else** **if** (list[i].get\_age() > 9 && list[i].get\_age() < 100) {
287. **for** (**int** i = 0; i < 7; i++) {
288. std::cout << " ";
289. }
290. }
291. **else** **if** (list[i].get\_age() > 99 && list[i].get\_age() < 1000) {
292. **for** (**int** i = 0; i < 6; i++) {
293. std::cout << " ";
294. }
295. }
297. std::cout << "|  " << list[i].get\_rating();
298. **if** (list[i].get\_rating() < 10) {
299. **for** (**int** i = 0; i < 13; i++) {
300. std::cout << " ";
301. }
302. }
303. **else** **if** (list[i].get\_rating() > 9 && list[i].get\_rating() < 100) {
304. **for** (**int** i = 0; i < 12; i++) {
305. std::cout << " ";
306. }
307. }
308. **else** **if** (list[i].get\_rating() > 99 && list[i].get\_rating() < 1000) {
309. **for** (**int** i = 0; i < 11; i++) {
310. std::cout << " ";
311. }
312. }
314. std::cout << std::endl;
315. }
317. std::cout << "+----+";
318. **for** (**int** i = 0; i < max+2; i++) {
319. std::cout << "-";
320. }
321. std::cout << "+-----------+-------------------" << std::endl;
323. /\*for (int i = 0; i < list.size(); i++) {
324. std::cout << list[i] << std::endl;
325. }\*/
326. }
328. **void** save\_to\_file(**const** std::vector <Student> &list, **const** TeamLeader &x) { //сохраняет данные в файл
329. std::cout << "Введите название файла: ";
330. std::string name\_of\_file;
331. std::cin >> name\_of\_file;
332. name\_of\_file += ".txt";
334. std::ofstream fout;
335. std::string sn, name, mn, str; **float** points\_; **int** years\_;
336. **int** n;
338. fout.open(name\_of\_file, std::ios::out);
340. **if** (!fout.is\_open()) { //если файл не открыт
341. std::cout << "Не удаётся открыть файл\n";
342. std::cin.get();std::cin.get();
343. }
345. **int** max = max\_lgth(list);
347. fout << "+----+";
348. **for** (**int** i = 0; i < max+2; i++) {
349. fout << "-";
350. }
351. fout << "+-----------+-------------------" << std::endl;
353. fout << "| №  | " << std::setw(max+1) << std::left << " ФИО студента";
354. fout << "|  Возраст  |  Средний балл  " << std::endl;
356. fout << "+----+";
357. **for** (**int** i = 0; i < max+2; i++) {
358. fout << "-";
359. }
360. fout << "+-----------+-------------------" << std::endl;
362. **for** (**int** i = 0; i < list.size(); i++) {
364. **if** (list.size() < 9) {
365. fout << "| " << i+1 << "  |";
366. }
367. **else** **if** ((list.size() >= 9) && (list.size() < 99)) {
368. fout << "| " << i+1 << " |";
369. }
370. fout << " " << list[i].get\_surname() << " " << list[i].get\_name() << " " << list[i].get\_midname() << " ";
371. **int** size = lgth(list[i]);
372. **for** (**int** i = 0; i < max - size+1;i++) {
373. fout << " ";
374. }
376. fout << "|  " << list[i].get\_age();
378. **if** (list[i].get\_age() < 10) {
379. **for** (**int** i = 0; i < 8; i++) {
380. fout << " ";
381. }
382. }
383. **else** **if** (list[i].get\_age() > 9 && list[i].get\_age() < 100) {
384. **for** (**int** i = 0; i < 7; i++) {
385. fout << " ";
386. }
387. }
388. **else** **if** (list[i].get\_age() > 99 && list[i].get\_age() < 1000) {
389. **for** (**int** i = 0; i < 6; i++) {
390. fout << " ";
391. }
392. }
394. fout << "|  " << list[i].get\_rating();
395. **if** (list[i].get\_rating() < 10) {
396. **for** (**int** i = 0; i < 13; i++) {
397. fout << " ";
398. }
399. }
400. **else** **if** (list[i].get\_rating() > 9 && list[i].get\_rating() < 100) {
401. **for** (**int** i = 0; i < 12; i++) {
402. fout << " ";
403. }
404. }
405. **else** **if** (list[i].get\_rating() > 99 && list[i].get\_rating() < 1000) {
406. **for** (**int** i = 0; i < 11; i++) {
407. fout << " ";
408. }
409. }
411. fout << std::endl;
412. }
414. fout << "+----+";
415. **for** (**int** i = 0; i < max+2; i++) {
416. fout << "-";
417. }
418. fout << "+-----------+-------------------" << std::endl;
420. fout << "\n\n";
422. fout << "|\_\_Староста\_группы\_\_|" << std::endl;
423. fout << "Фамилия: " << x.get\_surname() << std::endl;
424. fout << "Имя: " << x.get\_name() << std::endl;
425. fout << "Отчество: " << x.get\_midname() << std::endl;
426. fout << "Средний балл в рейтинге: " << x.get\_rating() << std::endl;
427. fout << "Возраст: " << x.get\_age() << std::endl;
428. fout << "Тел.: " << x.get\_tel() << std::endl;
430. fout.close(); //закрываем файл
432. std::cout << "Данные успешно записаны" << std::endl;
433. std::cin.get();std::cin.get();
435. }
437. **int** max\_lgth(**const** std::vector <Student> &list) {
438. **int** max = 0;
439. **for** (**int** i = 0; i < list.size(); i++) {
440. **int** size = list[i].get\_name().length() + list[i].get\_surname().length() + list[i].get\_midname().length() + 3;
441. **if** (size > max) {
442. max = size;
443. }
444. }
445. **return** max;
446. }
448. **int** lgth(**const** Student &obj) {
449. **int** size = obj.get\_name().length() + obj.get\_surname().length() + obj.get\_midname().length() + 3;
450. **return** size;
451. }

*ПРИЛОЖЕНИЕ Б. Глоссарий*

1. Наследование – свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским. Новый класс — потомком, наследником, дочерним или производным классом.
2. Классом является описываемая на языке терминологии исходного кода модель ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом.
3. Объект – сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса.
4. Конструктор – это особый метод, определяемый в классе, который вызывается при создании экземпляра класса. Конструкторы обычно используются для определения значений по умолчанию или для выполнения других операций по настройке объекта.
5. ПО – программное обеспечение.
6. Алгоритм – система последовательных операций (в соответствии с определенными правилами) для решения какой-нибудь задачи.
7. STL (библиотека стандартных шаблонов) – набор согласованных обобщенных алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.
8. Контейнер (в программировании) – тип, позволяющий инкапсулировать в себе объекты других типов.
9. Инкапсуляция – механизм, позволяющий скрывать поля и методы одного класса от других классов.