Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина: Основы алгоритмизации и программирования

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему:

**«СИСТЕМА УЧЕТА ЗАСЕЛЕНИЯ В ОБЩЕЖИТИЕ»**

Выполнил: гр.421703 Мацукевич З.А.

Проверила: Новицкая Л.И.

Минск 2025

**РЕФЕРАТ**

CИСТЕМА УЧЕТА ЗААСЕЛЕНИЯ В ОЩЕЖИТИЕ: курсовая работа / З. А. Мацукевич – Минск: БГУИР, 2025, – п.з. 45 с., 6 рис., 6 источников, блок-схема – 1 л. формата А4, листинг кода – 16 л. формата А4.

Объектом разработки является система для определения приоритета заселения в общежитие на языке программирования С/С++ в среде разработки Visual Studio Code.

Работа разделена на несколько глав:

* введение;
* структуры данных и работа с файлами;
* алгоритмы сортировки;
* алгоритмы поиска;
* пользовательские функции;
* описание работы программы;
* заключение.

Пояснительная записка представляет собой подробное описание реализации каждой части программы.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc199194068)

[1 Структуры и файлы 8](#_Toc199194069)

[1.1 Структура хранения информации о студенте 8](#_Toc199194070)

[1.2 Работа с файлами 8](#_Toc199194071)

[2 Алгоритмы сортировки 11](#_Toc199194072)

[2.1 Сортировка «выбором» 11](#_Toc199194073)

[2.2 Сортировка «вставками» 12](#_Toc199194074)

[2.3 Быстрая сортировка 12](#_Toc199194075)

[3 Алгоритмы поиска 15](#_Toc199194076)

[3.1 Линейный поиск 15](#_Toc199194077)

[3.2 Бинарный поиск 16](#_Toc199194078)

[4 Пользовательские функции 17](#_Toc199194079)

[4.1 Функции ввода/вывода 17](#_Toc199194080)

[4.2 Вспомогательные функции 19](#_Toc199194081)

[5 Описание работы программы 21](#_Toc199194082)

[Заключение 26](#_Toc199194083)

[Список использованных источников 27](#_Toc199194084)

[Приложение А (обязательное) Листинг кода 28](#_Toc199194085)

[Приложение Б (обязательное) Блок-схема работы программы 44](#_Toc199194086)

Введение

Актуальность темы курсовой работы обусловлена необходимостью оптимизации ресурсов студенческих общежитий, что является важной задачей для образовательных учреждений, стремящихся к эффективному управлению и рациональному распределению мест проживания. В современных условиях повышение прозрачности и справедливости предоставления мест в общежитии становится особенно значимым, поскольку оно напрямую влияет на комфорт и условия обучения студентов. При этом важно не только учитывать объективные показатели успеваемости и социальной активности учащихся, но и соблюдать социальные гарантии, предоставляя льготы нуждающимся.

Процесс предоставления мест в общежитии традиционно требует обработки больших объемов информации, включая заявки студентов, их академические достижения, участие в общественной жизни, а также социальные факторы, такие как уровень дохода семьи. Приоритетное заселение в общежитие получают студенты, обладающие высокой успеваемостью, активно участвующие в общественной деятельности и имеющие право на социальные льготы. Однако ручное распределение мест часто сопряжено с субъективностью и длительным сроком рассмотрения заявок. Автоматизация данного процесса позволит не только значительно сократить время ожидания результатов, но и исключить человеческий фактор, обеспечивая тем самым объективность, прозрачность и открытость системы.

Объектом исследования в данной работе выступает автоматизированная система учета заселения в общежитие, которая призвана решить проблему справедливого распределения мест с учетом установленных критериев.

Предмет исследования — алгоритм обработки массивов данных, включающий в себя сортировку, поиск и фильтрацию информации по заданным параметрам. Он является ключевым элементом системы, определяющим эффективность и точность работы программного обеспечения.

Основной целью данной работы является разработка программного обеспечения, способного автоматизировать процесс определения студентов, имеющих наибольший приоритет при распределении мест в общежитии. Для этого учитываются такие критерии, как средний балл, активность в общественной жизни и средний доход на одного члена семьи. В ходе выполнения работы также были предусмотрены дополнительные функциональные возможности системы, включая:

* добавление студента и информации о нем;
* вывод информации обо всех студентах в виде удобной таблицы;
* отображение детальной информации об одном студенте;
* редактирование данных о студенте с поиском по введенному имени;
* удаление данных о студенте с поиском по введенному имени;
* линейный поиск информации по имени студента;
* бинарный поиск данных по группе;
* быстрая сортировка списка студентов по имени;
* сортировка выбором по группе;
* сортировка вставками по среднему баллу;
* поиск студентов по группе с учетом их участия в общественной деятельности и академических успехов.

Для реализации работы была выбрана среда разработки Visual Studio Code (далее VS Code) — мощный и удобный инструмент, предоставляющий оптимизированный редактор кода с возможностью отладки, запуска задач и контроля версий. Разработанный компанией Microsoft, VS Code поддерживает работу на различных операционных системах, включая Windows, Linux и MacOS, и предоставляет разработчику удобный набор инструментов для быстрого цикла код-сборка-отладка.

В качестве основного языка программирования для данной работы был выбран С/С++. Эти низкоуровневые языки обеспечивают высокую скорость выполнения программ и широкие возможности для оптимизации алгоритмов. Простота синтаксиса языка C делает его доступным для изучения, а его расширенная версия — C++ — предлагает более богатый функционал. В рамках данной работы предполагается использование некоторых возможностей C++, однако основные функции и алгоритмы реализованы вручную, без использования готовых библиотек, что способствует лучшему пониманию принципов работы программных решений.

Таким образом, данная курсовая работа направлена на решение актуальной проблемы автоматизации учета студентов, заселяемых в общежитие, с целью повышения эффективности, прозрачности и удобства процесса распределения мест.

1 Структуры и файлы

Структура является объединением простых переменных. Эти переменные могут иметь различные типы: int, float и т. д. (как мы позже увидим, именно разнородностью типов переменных структуры отличаются от массивов, в которых все переменные должны иметь одинаковый тип). Переменные, входящие в состав структуры, называются полями структуры [1].

1.1 Структура хранения информации о студенте

В работе используется структура Student для хранения информации о студенте. У нужной структуры 5 полей:

* имя;
* номер группы;
* средний балл;
* наличие активности в общественной деятельности;
* средний доход в семье на человека.

Для хранения имени выделен массив типа char из сорока элементов name. Номер группы хранится и средний доход хранится в виде значения типа int group и income соответственно. Средний балл может быть дробным числом, так что необходимо поле score типа данных с плавающей запятой float. Для индикации участия в общественной деятельности достаточно поля activist типа bool. Итоговый вид структуры Student:

struct Student {

char name[40];

int group;

float score;

bool activist;

int income;

};

Для хранения данных будет использоваться массив структур Student, каждый элемент которого будет содержать соответствующую информацию.

1.2 Работа с файлами

На самом нижнем уровне файл представляет собой последовательность байтов, пронумерованных начиная с нуля. Файл имеет формат, иначе говоря, набор правил, определяющих смысл байтов. Например, если файл является текстовым, то первые четыре байта представляют собой первые четыре символа. С другой стороны, если файл хранит бинарное представление целых чисел, то первые четыре байта используются для бинарного представления первого целого числа. Формат по отношению к файлам на диске играет ту же роль, что и типы по отношению к объектам в основной памяти. Мы можем приписать битам, записанным в файле, определенный смысл тогда и только тогда, когда известен его формат [2]. Потому для хранения больших массивов данных выгоднее использовать бинарный файлы (расширение .bin). Именно в таком файле в данной работе и хранится неупорядоченный список всех студентов students.bin. Файлы такого типа абсолютно не читаемы для человека.

Также результат выполнения любой операции, помимо вывода в консоль, будет регистрироваться в файл log.txt. Данный файл нужен для просмотра результата выполнения программы человеком, так что использование бинарного файла неуместно, потому был выбран легко воспринимаемый человеком формат .txt.

Директория include/ содержит файл header.h — заголовочный файл в котором объявляются все функции о подключаются необходимые библиотеки (приложение В).

В директории src/ находятся файлы с исходным кодом. В header.cpp описана реализация всех функций (приложение Б), а main.cpp вывод меню и вызов ключевых функций (приложение А).

Языки C/C++ являются **компилируемыми**, что означает необходимость преобразования исходного кода в машинный код перед выполнением программы. В результате компиляции, объединяющую их с библиотеками в готовый исполняемый файл. Например, команда g++ main.cpp -o main собирает программу, создавая бинарный файл (main.exe в Windows или main без расширения в Linux), который можно запускать напрямую. Без этого этапа программа останется лишь текстовым кодом и не сможет быть выполнена компьютером.

Файловая структура работы имеет следующий вид:

coursework/

├─— include/

│ └─— header.h

├─— src/

│ ├─— header.cpp

│ └─— main.cpp

├─— log.txt

├─— main.exe

└─— students.bin

Для того чтобы можно было читать из файла или писать в файл, он должен быть предварительно открыт с помощью библиотечной функции fopen. Функция fopen получает внешнее имя типа log.txt или students.bin, после чего осуществляет некоторые организационные действия и "переговоры" с операционной системой и возвращает указатель, используемый в дальнейшем для доступа к файлу. Этот указатель, называемый указателем файла, ссылается на структуру, содержащую информацию о файле (адрес буфера, положение текущего символа в буфере, открыт файл на чтение или на запись, были ли ошибки при работе с файлом и не встретился ли конец файла). Определения, полученные из стандартной библиотеки <cstdio>, включают описание структуры, называемой FILE [3]. Для определения указателя требуется задать его описание:

FILE \*file = fopen(char \*name, char \*mode);

Есть несколько режимов работы mode, в данной работе использовались “at” для дополнения текстового файла и “wb”, “ab” и “rb” для перезаписи, дополнения и чтения бинарного файла соответственно.

Для чтения и записи в разные типы файла предусмотрены разные функции. Для записи в текстовый файл используется функция fprintf(FILE \*file, char \*format, …), где file — указатель на файл, а format определяет форматирование записи. Для записи в бинарный файл нужно использовать другую функцию fwrite(void \*struct, size\_t size, size\_t count, FILE \*file), где struct — указатель на структуру или ячейку в памяти, куда будут записаны данные, size — размер одного элемента в байтах, count — количество элементов для чтения, file — указатель на файл. Для чтения из бинарного файла используется fread() с аналогичным синтаксисом.

Функция fclose(FILE \*file) — обратная по отношению к fopen, она разрывает связь между файловым указателем и внешним именем (которая раньше была установлена с помощью fopen), освобождая тем самым этот указатель для других файлов. Так как в большинстве операционных систем количество одновременно открытых одной программой файлов ограничено, то файловые указатели, если они больше не нужны, лучше освобождать [3]. Если программа завершена правильно, то fclose вызывается автоматически.

2 Алгоритмы сортировки

Сортировка обеспечивает эффективную работу с информацией. Применение сортировки позволяет организовать данные в удобном для восприятия и обработки виде, что значительно упрощает выполнение повседневных задач администрации общежития. Существует бесчисленное множество алгоритмов сортировки, но в работе реализованы сортировка «вставками», «выбором» и алгоритм быстрой сортировки

2.1 Сортировка «выбором»

В этом алгоритме сортировки поиск осуществляется по всему массиву начиная с последнего поставленного на свое место элемента. Допустим необходим отсортировать массив по возрастанию, тогда ищем в массиве самый малый элемент и меняем его местами с первым элементом. Теперь ищем следующий элемент начиная со второго, после постановки его на место ищем следующий элемент начиная с третьего и так далее, пока весь массив не будет отсортирован. Алгоритм сортировки выбором по полю group структуры Student в данной работе:

for(int i = 0; i < stud\_count - 1; i++){

  int minIndex = i;

for(int j = i+1; j < stud\_count; j++)

if(studs[j].group < studs[minIndex].group)

minIndex = j;

if(minIndex != i) {

Student temp = studs[i];

studs[i] = studs[minIndex];

studs[minIndex] = temp;

}

}

Алгоритм пошагово описывается следующим образом:

* запускается цикл для i от 0 до stud\_count-1, где stud\_count — количество студентов в массиве;
* определяется переменная minIndex, хранящая индекс минимального элемента;
* запускается цикл для j от i+1 до stud\_count;
* сравнивается значение поля group у каждого j-ого элемента и minIndex-ого;
* если значение j-ого оказывается меньше, то minIndex присваивается новое значение, т.к. обнаружен элемент меньше, чем текущий minIndex;
* если после цикла для j minIndex изменился и больше не равен i, то элементы массива studs под индексами i и minIndex меняются местами;
* когда цикл для i закончится все элементы будут проверены, а исходный массив отсортирован.

В любом из случаев сложность такой сортировки составит O(n2).

2.2 Сортировка «вставками»

В этом методе сортировки просмотр элементов начинается со второго элемента массива. При этом просмотр производится от выбранного элемента key, к началу массива. Выбранный к упорядочиванию элемент массива сравнивается с элементами, лежащими левее его, эта часть массива уже отсортирована. Определяется индекс первого встретившегося элемента в отсортированной части массива, который меньше выбранного, при этом пока идет поиск этого элемента каждый неподходящий смещается на один элемент вправо, освобождая место для вставки key или смещения следующего значения. Все элементы с найденной позиции и до позиции выбранного элемента смещаются на один элемент вправо. В освободившуюся позицию найденного элемента записывается key. Переходим к следующему элементу и так до тех пор, пока не будут просмотрены все элементы массива. Если для некоторого элемента не находится в упорядоченной части слева элемента для перестановки, значит этот элемент уже находится на нужном месте и нужно переходить к анализу следующего элемента правее его. Алгоритм сортировки вставками по полю score структуры Student в данной работе:

for(int i = 1; i < stud\_count; i++){

Student key = studs[i];

int j;

for(j = i - 1; j >=0 && studs[j].score < key.score; j--)

studs[j+1] = studs[j];

studs[j+1] = key;

}

Алгоритм такой сортировки эффективнее сортировки «выбором», так как в лучшем случае сложность такой сортировки О(n) и лишь в худшем О(n2).

2.3 Быстрая сортировка

Метод Хоара быстрой сортировки (Quicksort), предложенный в 1960 г., построен на основе идеи разбиения множества на подмножества. Общая процедура заключается в выборе такого элемента, который разбивает множество на два подмножества так, что в одном (слева от делящего) меньшие, а в другом (справа от делящего) большие делящего. Этот процесс повторяется для каждого из полученных подмножеств, и так до тех пор, пока массив не будет отсортирован. Данный процесс по своей природе является рекурсивным. В качестве делящего элемента выбирается, например, срединный элемент множества [4].

В коде алгоритм быстрой сортировки применяется для сортировки списка студентов по ФИО:

void quicksort(Student \*studs, int l, int r) {

Student pivot = studs[(l+r) / 2], temp\_stud;

int lt = l, rt = r; // left temp & right temp

do {

while(nameComp(studs[lt].name, pivot.name) < 0) lt++;

while(nameComp(studs[rt].name, pivot.name) > 0) rt--;

if(lt <= rt){

temp\_stud = studs[lt];

studs[lt] = studs[rt];

studs[rt] = temp\_stud;

lt++;

rt--;

}

} while(lt <= rt);

if (l < rt) quicksort(studs, l, rt);

if (lt < r) quicksort(studs, lt, r);

}

Подробный разбор быстрой сортировки:

* функция принимает на вход массив studs, левую и правую границы сортировки l и r соответственно;
* определяется переменная pivot, хранящая студента, вокруг которого будет разделяться массив и temp\_stud, которая позже будет нужда для перестановки элементов местами;
* определяются переменные lt и rt, изначальна равные левой и правой границе сортировки соответственно;
* запускается цикл do-while, который выполняется один раз и пока lt<=rt;
* запускается два цикла while с начала массива и с конца в поиске слева элементов больших либо равных pivot, справа меньше либо равных. Для сравнения элементов создана функция nameComp, возвращающая 1, если первая строка больше, -1 — если вторая и 0, если они равны (подробнее работа функции описана в главе 4 пользовательские функции);
* когда элементы найдены они меняются местами с помощью переменной temp\_stud и индексы lt и rt смещаются на следующую для них позицию;
* в результате цикла получаем массив, в котором элементы слева от pivot меньше его, а справа – больше;
* если левая граница левее, чем правый индекс rt, то вызываем функцию для этого участка, если правая граница правее, чем левый индекс lt, то вызываем сортировку и для этого участка. В противном случае это значит, что участок из одного элемента, значит уже отсортирован.
* так как сортировка производится по имени, а в имени могут быть пробелы, то был написан компаратор для сравнения строк содержащих ФИО, работа которого будет объяснена в 4-ой главе. Средняя скорость такой сортировки O(n\*log2(n)), в худшем случае такая сортировка имеет сложность O(n2), но для этого массив должен быть отсортирован заранее, иметь большое количество одинаковых элементов или в качестве pivot изначально должен быть выбран элемент наибольшего или наименьшего значения.

Так как сортировка «выбором» всегда имеет сложность O(n2), а «вставками» приближается к O(n) только когда массив почти отсортирован, что в случае со списком из ФИО студентов и любых других случайных массивов данных весьма маловероятно. Потому алгоритм Хоара является одним из самых эффективных из этих трех и вообще и его выбор для этой задачи оправдан.

3 Алгоритмы поиска

В современных информационных системах, особенно в системах учета и управления данными, эффективный поиск является одной из ключевых задач. Конечно, как и алгоритмов сортировки, алгоритмов поиска множество, но все они делятся на две категории: поиск в упорядоченно массиве данных и в неупорядоченном. В работе реализовано по одному типу поиска из каждой категории. Линейный поиск по ФИО студента для неупорядоченных данных и бинарный по номеру группы для упорядоченных.

3.1 Линейный поиск

Линейный поиск — это простой алгоритм поиска элемента в неупорядоченном массиве данных. Суть его заключается в том, что он последовательно сравнивает каждый элемент массива с искомым значением. Таким образом в худшем случае сложность алгоритма O(n).

В программе линейный поиск осуществляется по ФИО студента, так что для сравнения вызывается функция nameComp и в случае возвращения функцией нулевого значения элемент считается найденным, и информация выводится в консоль и логируется в одноименный файл при помощи нескольких самописных функций. Также учтено, что студентов с одинаковым ФИО хоть и редко, но может быть несколько, так что алгоритм не прерывается, при нахождении нужного студента, а продолжает поиск:

bool success = false;

for(int i = 0; i < stud\_count; i++){

if(nameComp(studs[i].name, findName) == 0){

if(!success){

txt = fopen("log.txt", "at");

fprintf(txt, "--LINEAR SEARCH BY NAME--\n");

txtTable(txt);

printTable();

}

printStudent(&studs[i]);

txtOutputStudent(txt, studs[i]);

success = true;

}

}

В результате в консоль выводится информация обо всех студентах с искомым ФИО, а переменной success присваивается положительное значение, в противном случае переменная success остается отрицательным и результат поиска обрабатывается особым образом.

3.2 Бинарный поиск

Бинарный поиск — алгоритм поиска в отсортированном массиве. Бинарным поиск назван потому, что разделяет список на две части по какому-то среднему элементу, а затем выполняет сравнение среднего с нужным значением find\_group. Исходя из результатов сравнения, выбирается один из двух списков, в котором будет продолжаться поиск. Начинается поиск со списка с некой длиной len, равной количеству элементов массива, начинающегося с заданного итератора first. Нужен цикл, чтобы продолжать исполнение, пока список не окажется пустым, то есть while(length > 0). Определяется середина half=length/2, выполняется сравнение и обновляется текущий список, что происходит при помощи изменения first и length [6]. Вот так:

int binarySearch(Student \*studs, int stud\_count,

int find\_group)

{

int start = 0;

int len = stud\_count;

while(len > 0) {

int half = len / 2;

if (studs[start + half].group < find\_group) {

start += half + 1;

len -= half + 1;

}

else

len = half;

}

if(start < stud\_count && studs[start].group == find\_group)

return start;

else

return -1;

}

Функция возвращает индекс первого вхождения студента из группы, номер которой равен искомой, чтобы можно было последовательно вывести всех студентов этой группы. В случае не нахождения студентов такой группы функция вернет -1, для дальнейшей обработки вывода. Сложность такого поиска O(log(n)).

4 Пользовательские функции

При написании кода существует множество негласных правил «хорошего тона», одним из которых является запрет на повторение одного и того же куска кода множество раз, для этого создаются функции. С помощью них также повышается читаемость кода, ведь по названию функции можно понять, что происходит в этот момент и не нужно разбираться в логике автора. Так и в этой курсовой работе помимо главных функций, выполнение которых прописано в условии, существует еще два множества: функции ввода/вывода и вспомогательные.

4.1 Функции ввода/вывода

Важной составляющей любого проекта является пользовательский интерфейс, потому красивый ввод и вывод данных в этой работе описан особым образом. Всего 12 функций для организации качественного опыта пользователя и корректной работы программы.

Функция inputNewGroup запрашивает у пользователя шестизначный номер группы, в случае некорректного ввода (ввода числа не шести знаков или не числа) выводит сообщение об ошибке и запрашивает ввод снова;

Функция inputNewScore запрашивает средний балл студента в виде числа от 0 до 10, возможно дробного. В случае некорректного ввода выводит сообщение об ошибке и запрашивает ввод снова.

Функция inputNewActivist спрашивает у пользователя является ли студент, информация о котором вводится, активистом и требует ввести 1, если это так, или 0, в противном случае. При некорректном вводе информирует об ошибке и запрашивает повторный ввод.

Функция inputNewIncome запрашивает средний доход на человека в семье этого студента в виде целочисленного положительного числа. В случае некорректного ввода информирует об ошибке и запрашивает ввод снова.

Функция inputStudentInfo получает на вход указатель на структуру Student и запрашивает у пользователя новое имя студента в виде строки до 40 символов. После чего вызывает 4 описаные выше функции для получения информации об учебной группе, среднем балле, активности в общественной деятельности и среднем доходе на члена семьи. После вызывает функции printTable и printStudent, передавая в последнюю указатель на структуру, которая только что заполнялась. Принцип работы последних функций будет описан ниже.

Функция printTable выводит в консоль шапку таблицы, выделяя 40 символов для столбца имен, 10 для группы, 10 для ср. балла, 15 для информации об активности студента и 20 для среднего дохода в семье студента.

Функция printStudent принимает на вход указатель на структуру Student и выводит информацию о студенте в формате, аналогичном шапке таблицы (функция printTable). При помощи условного выражения обрабатывается булевое поле activist и выводится “yes” при положительном значении и “no” в противном случае.

Функция getStudentArray принимает на вход информацию о количестве студентов stud\_count в бинарном файле students.bin, открывает его в режиме “rb” для бинарного чтения и обрабатывает неудачную попытку открыть файл с выводом в консоль и логи соответствующего сообщения. Затем создает динамический массив структуры Student соответствующего размера и при помощи функции fread и цикла for для i от 0 до stud\_count считывает информацию из файла, после чего возвращает указатель на первый элемент массива.

Функция writeStudentArray обратна предыдущей. Принимая на вход указатель на первый элемент массива структуры Student и количество в нем элементов stud\_count открывает файл students.bin в режиме “wb” для бинарной перезаписи (очищает файл и записывает новые данные, в случае не нахождения файла с таким названием автоматически создается новый) и в ходе цикла for для i от 0 до stud\_count производит запись структур Student из полученного массива.

Функция getStudentCount существует для вычисления количества элементов в бинарном файле students.bin. Сначала открывает файл в режиме “rb” и обрабатывает соответствующую ошибку. При помощи функции fseek(file, 0, SEKK\_END) устанавливается указатель в файле в конец файла (SEEK\_END) и переменной file\_size присваевается позиция указателя при помощи ftell(file). Так как в бинарном файле хранятся только струкутры Student, то при делении полученного количества байт на количество байт, занимаемого одной структурой Student можно получить количество структур, хранящихся в файле. Именно это число и возвращает эта функция.

Функция logTable принимает на вход указатель на файл логирования и записывает в файл шапку таблицы аналогичную из функции printable при помощи функции fprintf.

Функция logOutputStudent принимает на вход указатель на файл логирования и ссылку на структуру Student. При помощи fprintf в файл записывается информация о студенте отформатированная аналогичным шапке таблицы образом.

Функция logOutpuArray принимает на вход указатель на файл логирования, указатель на первый элемент массива структур Student и количество элементов в массиве stud\_count. В ходе цикла for для i от 0 до stud\_count при помощи функции logOutputStudent записывает в файл информацию о каждом студенте из массива.

4.2 Вспомогательные функции

Разработка сложных программных систем требует не только реализации основных алгоритмов, но и тщательной проработки вспомогательных механизмов, обеспечивающих гибкость и надежность кода. В данной программе вспомогательные функции выступают в роли своеобразного "фундамента", на котором строится вся логика работы. Их можно сравнить с хорошо отлаженными инструментами в руках мастера – каждый выполняет свою узкоспециализированную задачу, но вместе они образуют слаженный механизм.

Функция findStudentByName плучая на вход указатель на первый элемент массива структур Student, строку name (в виде указателя на массив char), поиск которого будет осуществляться и количество элементов в массиве. Функция запускает линейный поиск по массиву сравнивая значения компаратором nameComp. При успешном поиске функция тут же возвращает первое вхождение искомого имени, в противном случае возвращает nullptr.

Функция binarySearchByGroup осуществляет бинарный поиск, алгоритм работы которого описан в подглаве 3.2. Функция возвращает указатель на первое вхождение студента искомой группы и -1 в случае его отсутствия в списке.

Принцип работы функций selectionSortByGroup, insertionSortByAvScor и QuickSortByName описаны выше, в подглавах 2.1, 2.2 и 2.3 в соответствующем порядке.

Функция-компаратор nameComp получает на вход два массива char и возвращает 1 в случае, если первая строка больше, -1 в случае, если вторая больше, и 0, если строки равны. Сразу выполняется проверка строк на пустоту. Если какая-то из полученных строк пуста, то вторая считается большей, если обе строки равны, то они принимаются за равные. Далее используя свойство массивов С/С++, элементы которых находятся в памяти рядом, а указателем на массив является указатель на первый его элемент, можно не обращаться к каждому элементу по индексу при переборе, а сдвигать указатель, увеличивая указатель на 1. Таким образом при помощи двух циклов while выполняется сдвиг «начала» массива на некоторое количество бит для игнорирования пробелов, с которых может начаться строка. Далее в бесконечном цикле while(true) выполняется проверка хотя бы одной из строк на нулевой символ ‘\0’, которым должна заканчиваться строка, и возвращает соответствующее значение. Если ни одна из строк не закончилась, то выполняется сравнение символов по таблице ASCII. Если какой-то из символов оказывается больше, функция возвращает соответствующее значение, в противном случае сдвигает указатели на строки еще на 1 значение и переходит к следующей итерации цикла.

Функция comparePriority нужна для выполнения ключевой задачи формирования списка приоритета на заселение в общежитие. Она принимает на вход указатели на двух студентов и значение минимального дохода min\_salary. Сразу же создается по булевой переменной для каждого студента, хранящие положительное значение, если доход на члена семьи студента меньше min\_salary\*2, и отрицательное в противном случае. Далее результат булевых переменных сверяется и в случае несоответствия приоритет отдается студенту с положительной булевой переменной. Затем сравнивается средний балл каждого из студентов и в случае несоответствия, приоритет отдается студенту с большим средним баллом. И наконец если средний балл был равен, то приоритет отдается студенту, который принимал активное участие в общественной жизни. Если и эти показатели равны, то компаратор отдаст приоритет второму, если оба активисты и первому, если оба не активисты.

Таким образом, тщательно спроектированные вспомогательные функции не просто упрощают реализацию основных алгоритмов – они создают прочный каркас, который обеспечивает стабильность работы всей системы. Их разработка потребовала глубокого анализа предметной области и внимания к деталям, но полученный результат полностью оправдал вложенные усилия. Такой подход к организации кода особенно важен в системах обработки данных, где корректность и предсказуемость работы являются критически важными требованиями.

5 Описание работы программы

При запуске программы в консоль выводится меню (рис. 5.1) а в логи записывается строка «--START THE PROGRAM--».

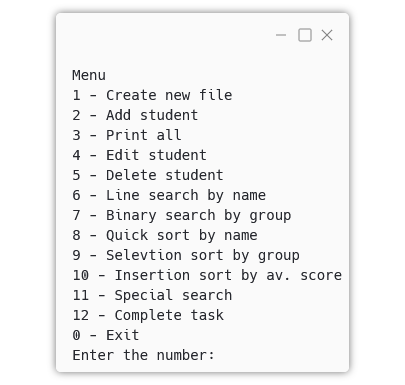


Рисунок 5.1 – Меню.

При некорректном вводе пункта меню программа проинформирует об ошибке, выводит в логи «--UNKNOWN OPERATION--», дублирует все пункты меню и запрашивает повторный ввод. При выборе пункта 1 перезаписывается старый бинарный файл. Тем самым вся информация о всех ранее записанных студентах стирается. В консоль выводится сообщение об успешном создании файла а в логи записывается выбранная операция   
«--CREATE FILE--» и подпись «New file was created».

Вторая опция в меню — это добавление информации об одном новом студенте. После ввода цифры 2 в консоль программа запрашивает данные о новом элементе. Сначала ФИО. Предполагается что ФИО может поместиться в 40 символов, потому для этого выделена строка именно такой длины. Т.к. имя может быть абсолютно любым, то никаких проверок на этом этапе не выполняется. Далее необходимо ввести шестизначный номер группы, о чем программа информирует. При вводе отрицательного числа, числа больше шести знаков или любого набора символов программа информирует о некорректном вводе и запрашивает повторный. Далее идет средний балл. Он может быть от 0 до 10, так что программа ожидает дробное или целое число в этом диапазоне, в противном случае запрашивает повторный ввод. Теперь нужно указать является ли студент активистом. Для этого вводится 1 если да, и 0 в противном случае. Последнее поле, которое необходимо заполнить — это средний доход на члена семьи. Предполагается, что это целое положительное число. При некорректном вводе программа уведомит об ошибке и запросит повторный ввод (рис. 5.2). Когда все поля будут заполнены программа выведет шапку таблицы, для обозначения каждого поля студента, в следующей строке в соответствующем столбце будет информация об добавленном студенте, в последней строке «Student was added» (рис. 5.3) и аналогичным образом информация логируется с заголовком «--ADD STUDENT--».

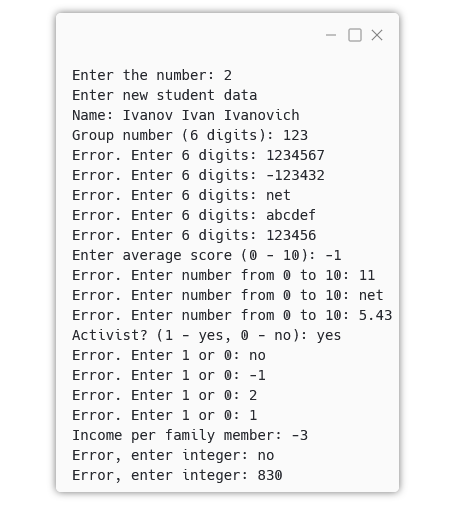


Рисунок 5.2 – Ввод информации о студенте.



Рисунок 5.3 – Вывод информации о студенте.

Третья операция в меню выводит всю информацию о всех студентах, имеющихся в бинарном файле в порядке, в котором они были добавлены (рис 5.4). В логи с заголовком «--PRINT ALL--» консольный вывод дублируется.

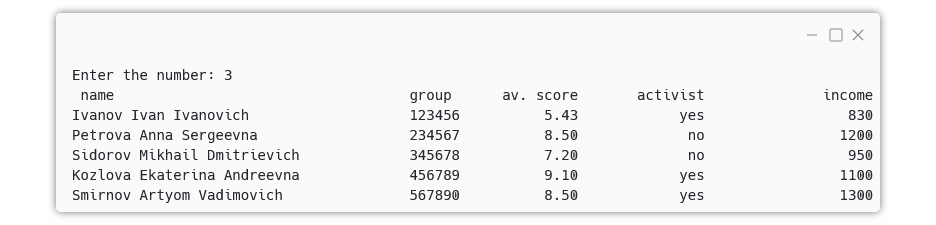


Рисунок 5.4 – Вывод информации обо всех студентах.

При выполнении четвертого пункта меню «Edit student» у пользователя запрашивается имя студента, информацию о котором нужно изменить. После производится поиск. Если студента с таким именем нет в бинарном файле, то пользователь информируется об этом (рис. 5.5), а информация логируется с соответствующим заголовком. Если такой студент имеется в списке, то у пользователя запрашивается новая информация в виде, аналогичным в пункте 2 «Add student» с соответствующими проверками (рис. 5.2). После чего в логи записывается сообщение об успешной перезаписи, старыые и новые данные об этом студенте в виде таблицы (рис. 5.4).

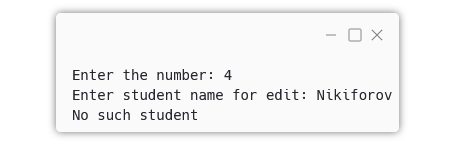


Рисунок 5.5 – Студент не найден.

В пункте пять «Delete student» программа аналогично четвертому пункту «Edit student» запрашивает имя студента и выводит аналогичное сообщение при ненахождении его в списке. При нахождении такого студента в бинарном файле программа стирает о нем информацию и выводит соответствующее сообщение в консоль и логи.

Пункт 6 «Line search by name» в меню запрашивает ввод имени студента и выводит всю информацию о нем как в случае с пунктом 3 «Print all» (рис. 5.4), с той лишь разницей, что студент один (если в списке имеются полный тезка, то будет выведена информация о всех студентах с таким именем). В зависимости от результата поиска в логи записывается аналогичный консольному вывод с заголовком «--LINEAR SEARCH BY NAME--».

В пункте 7 «Binary search by group» программа запросит номер группы студентов, которых нужно найти. Информация о них будет выведена аналогично пункту «Print all» (рис 5.4). Если таких студентов нет программа проинформирует об этом пользователя. В зависимости от результатов поиска в логи запишется сообщение, аналогичное консольному выводу с заголовком «--BINARY SEARCH BY GROUP--».

В пункте 8 «Quick sort by name» выполняется быстрая сортировка по имени студентов в алфавитном порядке. Формат вывода аналогичен пункту «Print all» (рис 5.4). В логи выводится заголовок   
«--QUICK SORT BY NAME--» и соответствующее сообщение: список пуст или таблица с информацией о студентах в алфавитном порядке. Если в списке студентов есть полные тезки, то они в конечном итоге будут стоять рядом в случайном порядке, без дополнительной сортировки по номеру группы, среднему баллу и т.д.

Сортировка выбором по номеру группы происходит при выборе пункта 9 «Selection sort by group» меню. Она выводит информацию о всех студентов в бинарном файле в порядке возрастания их номера группы. Если студентов в одной группе несколько, то они стоят в списке рядом, но никак дополнительно не отсортированы. После всех операций логируется заголовок «--SELECTION SORT BY GROUP--» и выводится список студентов в нужном порядке либо сообщение об ошибке.

В порядке убывания среднего балла студенты сортируются при вызове пункта меню 10 «Insertion sort by av. score». В результате выводится список студентов в соответствующем порядке, а логируется вывод с заголовком «--INSERTION SORT BY AV. SCORE--» и надписью о пустом бинарном файле или списке студентов в отсортированном виде.

Пункт меню 11 «Special search» выполняет поиск по признаку студентов в списке. Для этого у пользователя запрашивается номер студенческой группы и необходимый средний балл. При вводе значений выполняются все необходимые проверки, как при вызове пункта меню 2 «Add student» (рис. 5.2). В результате такого поиска выводится список о студентах учащихся введенной пользователем группы, средний балл которых не ниже заданного. Также эти студенты должны проявить активность в общественной деятельности. Информация об этих студентах логируется с заголовком «--SPECIAL SEARCH--». В случае отсутствия подходящих студентов программа также выводит информацию об этом как в логи, так и в консоль.

Последний пункт меню 12 «Complete task» — есть суть данной курсовой работы. В процессе выполнения данной операции список студентов обрабатывается таким образом, что формируется новый список, в котором студенты расположены в порядке приоритета на заселение в общежитие. В первую очередь список делится на студентов, чей средний доход на члена семьи меньше удвоенного значения минимального дохода, которое программа запрашивает у пользователя с проверками на отрицательность числа и вообще, является ли введенное значение числом (рис. 5.6). Далее каждая часть получившегося списка сортируется по среднему баллу студентов в порядке убывания, чтобы наиболее успешные оказались вверху списка. Если вдруг получается так, что несколько студентов из одной категории по среднему доходу на члена семьи имеет идентичный средний балл, то приоритет отдается студентам, проявившим активное участие в общественной деятельности. В итоге пользователю выводится отсортированный по этим правилам список или сообщение о пустом бинарном файле. В логи с заголовком «--QUEUE TO DORM--» и дублированием консольного вывода, будь то сообщение об ошибке или готовый список.

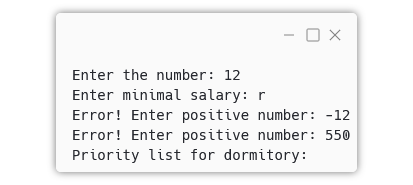


Рисунок 5.6 – Ввод минимального дохода на члена семьи.

Также при выполнении каждого пункта меню, где выполняется чтение бинарного файла (пункты 3-12), выполняется также проверка на его пустоту. При выполнении, например, операции 3 «Print all» и пустом бинарном файле в консоль и логи выведется сообщение о его пустоте. В логах также будет записано какая операция вызвала эту проверку.

При завершении работы программы в логи записывается сообщение «--CLOSE THE PROGRAM--».

Принцип работы программы представлен также в виде блок-схемы (приложение Б).

Заключение

В ходе выполнения курсовой работы была успешно разработана программа, автоматизирующая процесс распределения мест в студенческом общежитии. Актуальность данной разработки подтверждается современными требованиями к прозрачности и эффективности управления ресурсами учебных заведений. Программа учитывает ключевые факторы, влияющие на приоритетность заселения: академическую успеваемость, общественную активность и материальное положение студентов, что позволяет соблюдать принципы социальной справедливости при распределении ограниченного количества мест.

Разработано программное обеспечение на языке C/C++ в среде Visual Studio Code. В его основе лежат самостоятельно реализованные алгоритмы сортировки и поиска, обеспечивающие быструю обработку информации о студентах. Особое внимание уделено удобству работы с системой — программа предоставляет интуитивно понятный интерфейс для ввода и редактирования данных, а также формирования отчетов.

Практическая значимость работы заключается в существенном сокращении временных затрат на обработку заявок и минимизации влияния человеческого фактора при принятии решений. Дальнейшее развитие работы может включать расширение функциональности за счет интеграции с базами данных учебного заведения и реализации веб-интерфейса для более удобного взаимодействия пользователей с системой.

Список использованных источников

[1] Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4 изд. – СПб.: Питер, 2004. – 923 с.

[2] Алгоритмы, принципы и практика с использованием С++, 2-е изд.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2016. – 1328с.

[3] Б. Керниган, Д. Ритчи. Язык программирования Си. 2-е изд: «Невский Диалект», 2003. – 355с.

[4] Луцик, Ю. А. Основы алгоритмизации и программирования : язык Си : учебно-метод. пособие / Ю. А. Луцик, А. М. Ковальчук, Е. А. Сасин. – Минск : БГУИР, 2015. – 170с. : ил.

[5] METANIT.COM [Электронный ресурс]. – Режим доступа: https://metanit.com/cpp/

[6] Хабр [Электронный ресурс]. – Сообщество IT-специалистов. Режим доступа: https://habr.com/ru/

Приложение А  
(обязательное)  
Листинг кода

**Файл “main.cpp”:**

1. #include "../include/header.h"
2. using namespace std;
3. int main() {
4. setlocale(LC\_ALL, "ru");
5. FILE \*txt = fopen("log.txt", "at");
6. fprintf(txt, "--START THE PROGRAM--\n\n");
7. fclose(txt);
8. int choise;
9. bool end\_program = false;
11. do {
12. cout << "Menu\n"
13. "1 - Create new file\n"
14. "2 - Add student\n"
15. "3 - Print all\n"
16. "4 - Edit student\n"
17. "5 - Delete student\n"
18. "6 - Line search by name\n"
19. "7 - Binary search by group\n"
20. "8 - Quick sort by name\n"
21. "9 - Selevtion sort by group\n"
22. "10 - Insertion sort by av. score\n"
23. "11 - Special search\n"
24. "12 - Complete task\n"
25. "0 - Exit\n";
26. cout << "Enter the number: ";
27. cin >> choise;
28. cin.ignore();
30. switch (choise) {
31. case 1:
32. createFile();
33. break;
34. case 2:
35. addStudent();
36. break;
37. case 3:
38. printAll();
39. break;
40. case 4:
41. editStudent();
42. break;
43. case 5:
44. deleteStudent();
45. break;
46. case 6:
47. searchName();
48. break;

**Продолжение приложения А**

1. case 7:
2. searchGroup();
3. break;
4. case 8:
5. sortName();
6. break;
7. case 9:
8. sortGroup();
9. break;
10. case 10:
11. sortAvScore();
12. break;
13. case 11:
14. specialSearch();
15. break;
16. case 12:
17. getStats();
18. break;
19. case 0:
20. end\_program = true;
21. txt = fopen("log.txt", "at");
22. fprintf(txt, "--CLOSE THE PROGRAM--\n\n");
23. fclose(txt);
24. break;
25. default:
26. cout << "Unknown operation!\n";
27. txt = fopen("log.txt", "at");
28. fprintf(txt, "--UNKNOWN OPERATION--\n\n");
29. fclose(txt);
30. break;
31. }
32. } while (!end\_program);
34. return 0;
35. }

**Файл “header.cpp”:**

1. #include "../include/header.h"
2. using namespace std;
3. void createFile() {
4. FILE\* file = fopen("students.bin", "wb");
5. fclose(file);
6. cout << "File was created\n\n";
7. FILE \*txt = fopen("log.txt", "at");
8. fprintf(txt, "--CREATE FILE--\n"
9. "New file was created\n\n");
10. fclose(txt);
11. }
12. void addStudent() {
13. FILE\* file = fopen("students.bin", "ab");
14. if(!file) {

**Продолжение приложения А**

1. cout << "File open error!\n\n";
2. FILE \*txt = fopen("log.txt", "at");
3. fprintf(txt, "--FILE OPEN ERROR--\n\n");
4. fclose(txt);
5. return;
6. }
7. Student new\_stud;
8. cout << "Enter new student data\n";
9. inputStudentInfo(&new\_stud);
10. fwrite(&new\_stud, sizeof(Student), 1, file);
11. fclose(file);
13. cout << "Student was added\n\n";
14. FILE \*txt = fopen("log.txt", "at");
15. fprintf(txt, "--ADD STUDENT--\n");
16. logTable(txt);
17. logOutputStudent(txt, new\_stud);
18. fprintf(txt, "\n");
19. fclose(txt);
20. }
21. int inputNewGroup() {
22. int group;
23. cout << "Group number (6 digits): ";
24. while (true) {
25. cin >> group;
26. if (cin.fail() || group < 100000
27. || 999999 < group) {
28. cout << "Error. Enter 6 digits: ";
29. cin.clear();
30. cin.ignore(numeric\_limits<streamsize>::max(),
31. '\n');
32. } else
33. return group;
34. }
35. }
36. float inputNewScore() {
37. float score;
38. cout << "Enter average score (0 - 10): ";
39. while (true) {
40. cin >> score;
41. if (cin.fail() || score < 0.0 || score > 10.0) {
42. cout << "Error. Enter number from 0 to 10: ";
43. cin.clear();
44. cin.ignore(numeric\_limits<streamsize>::max(),
45. '\n');
46. } else
47. return score;
48. }
49. }
50. bool inputNewActivist() {
51. int activist;
52. cout << "Activist? (1 - yes, 0 - no): ";

**Продолжение приложения А**

1. while (true) {
2. cin >> activist;
3. if (cin.fail() || (activist != 1 && activist!=0)){
4. cout << "Error. Enter 1 or 0: ";
5. cin.clear();
6. cin.ignore(numeric\_limits<streamsize>::max(),
7. '\n');
8. } else
9. return activist == 1;
10. }
11. }
12. int inputNewIncome() {
13. int income;
14. cout << "Income per family member: ";
15. while (true) {
16. cin >> income;
17. if(cin.fail() || income < 0) {
18. cout << "Error, enter integer: ";
19. cin.clear();
20. cin.ignore(numeric\_limits<streamsize>::max(),
21. '\n');
22. } else
23. return income;
24. }
25. }
26. void printStudent(Student\* stud) {
27. cout << setw(40) << left << stud->name
28. << setw(10) << left << stud->group
29. << setw(10) << right << fixed
30. << setprecision(2) << stud->score
31. << setw(15) << right
32. << (stud->activist ? "yes" : "no")
33. << setw(20) << right
34. << stud->income << endl;
35. }
36. void printStudentArray(Student \*studs,
37. int stud\_count) {
38. for(int i = 0; i < stud\_count; i++)
39. printStudent(&studs[i]);
40. }
41. void printTable() {
42. cout << setw(40) << left << " name"
43. << setw(10) << left << "group"
44. << setw(10) << right << "av. score"
45. << setw(15) << right << "activist"
46. << setw(20) << right << "income" << endl;
47. }
48. void inputStudentInfo(Student\* stud) {
49. cout << "Name: ";
50. cin.getline(stud->name, 40);

**Продолжение приложения А**

1. stud->group = inputNewGroup();
2. stud->score = inputNewScore();
3. stud->activist = inputNewActivist();
4. stud->income = inputNewIncome();
6. printTable();
7. printStudent(stud);
8. }
9. void printAll() {
10. FILE \*file = fopen("students.bin", "rb");
11. if(!file) {
12. cout << "File open error!\n\n";
13. FILE \*txt = fopen("log.txt", "at");
14. fprintf(txt, "--FILE OPEN ERROR--\n\n");
15. fclose(txt);
16. return;
17. }
18. if(getStudentCount() == 0) {
19. cout << "Database is empty\n";
20. FILE \*txt = fopen("log.txt", "at");
21. fprintf(txt, "--PRINT ALL--\n"
22. "Database is empty\n\n");
23. fclose(txt);
24. return;
25. }
26. Student stud;
27. FILE \*txt = fopen("log.txt", "at");
28. printTable();
29. fprintf(txt, "--PRINT ALL--\n");
30. logTable(txt);
31. while(fread(&stud, sizeof(Student), 1, file)) {
32. printStudent(&stud);
33. logOutputStudent(txt, stud);
34. }
35. cout << endl;
36. fprintf(txt, "\n");
37. fclose(file);
38. fclose(txt);
39. }
40. Student \*getStudentArray(int stud\_count) {
41. FILE \*file = fopen("students.bin", "rb");
42. if(!file) {
43. cout << "File open error!\n\n";
44. FILE \*txt = fopen("log.txt", "at");
45. fprintf(txt, "--FILE OPEN ERROR--\n\n");
46. fclose(txt);
47. return 0;
48. }
49. Student \*studs = new Student[stud\_count];
50. for(int i = 0; i < stud\_count; i++)

**Продолжение приложения А**

1. fread(&studs[i], sizeof(Student), 1, file);
3. return studs;
4. }
5. void writeStudentArray(Student \*studs, int stud\_count){
6. FILE \*file = fopen("students.bin", "wb");
7. for(int i = 0; i < stud\_count; i++)
8. fwrite(&studs[i], sizeof(Student), 1, file);
9. fclose(file);
10. }
11. int nameComp(char \*first, char \*second) {
12. if (!first || !second)
13. return (!first && !second) ? 0 : ((!first)?-1:1);
14. while (\*first == ' ') first++;
15. while (\*second == ' ') second++;
16. while (true) {
17. if (\*first == '\0' || \*second == '\0')
18. return (\*first == '\0' && \*second == '\0') ? 0
19. : ((\*first == '\0') ? -1 : 1);
21. if (\*first != \*second) {
22. return (\*first > \*second) ? 1 : -1;
23. }
24. first++;
25. second++;
26. }
27. }
28. void editStudent() {
29. int stud\_count = getStudentCount();
30. if (stud\_count == 0) {
31. cout << "Database is empty\n";
32. FILE \*txt = fopen("log.txt", "at");
33. fprintf(txt, "--EDIT STUDENT--\n"
34. "Database is empty\n\n");
35. fclose(txt);
36. return;
37. }
38. Student \*studs = getStudentArray(stud\_count);
39. char editName[40];
40. cout << "Enter student name for edit: ";
41. cin.getline(editName, 40);
42. Student \*studentForEdit = findStudentByName(studs,
43. editName, stud\_count);
44. FILE \*txt = fopen("log.txt", "at");

**Продолжение приложения А**

1. if (studentForEdit != nullptr) {
2. cout << "Current student data:\n";
3. printTable();
4. printStudent(studentForEdit);
6. fprintf(txt, "--EDIT STUDENT--\n"
7. "Edited student:\n");
8. logTable(txt);
9. logOutputStudent(txt, \*studentForEdit);
10. cout << "\nEnter new info:\n";
11. inputStudentInfo(studentForEdit);
13. logOutputStudent(txt, \*studentForEdit);
14. fprintf(txt, "\n");
15. writeStudentArray(studs, stud\_count);
16. cout << "Changes were saved\n\n";
17. } else {
18. cout << "No such student\n\n";
19. fprintf(txt, "--EDIT STUDENT--\n"
20. "Student %s not found\n\n", editName);
21. }
22. fclose(txt);
23. delete[] studs;
24. }
25. int getStudentCount() {
26. FILE \*file = fopen("students.bin", "rb");
27. if(!file) {
28. cout << "File open error!\n\n";
29. FILE \*txt = fopen("log.txt", "at");
30. fprintf(txt, "--FILE OPEN ERROR--\n\n");
31. fclose(txt);
32. return 0;
33. }
34. fseek(file, 0, SEEK\_END);
35. long file\_size = ftell(file);
37. fclose(file);
38. return file\_size / sizeof(Student);
39. }
40. Student \*findStudentByName(Student \*studs,
41. char \*name, int stud\_count){
42. for(int i = 0; i < stud\_count; i++)
43. if(nameComp(studs[i].name, name) == 0)
44. return &studs[i];
45. return nullptr;
46. }
47. void deleteStudent() {
48. int stud\_count = getStudentCount();
49. if (stud\_count == 0) {

**Продолжение приложения А**

1. cout << "Database is empty\n";
2. FILE \*txt = fopen("log.txt", "at");
3. fprintf(txt, "--DELETE STUDENT--\n"
4. "Database is empty\n\n");
5. fclose(txt);
6. return;
7. }
8. Student \*studs = getStudentArray(stud\_count);
9. char deleteName[40];
10. bool success = false;
11. cout << "Enter student name for delete: ";
12. cin.getline(deleteName, 40);
13. FILE \*file = fopen("students.bin", "wb");
14. for(int i = 0; i < stud\_count; i++) {
15. if(nameComp(studs[i].name, deleteName) == 0) {
16. success = true;
17. continue;
18. }
19. fwrite(&studs[i], sizeof(Student), 1, file);
20. }
21. fclose(file);
22. delete[] studs;
24. if(success) {
25. FILE \*txt = fopen("log.txt", "at");
26. fprintf(txt, "--DELETE STUDENT--\n"
27. "Student was deleted\n\n");
28. fclose(txt);
29. cout << "Changes was saved\n\n";
30. }
31. else {
32. cout << "No such student\n\n";
33. FILE \*txt = fopen("log.txt", "at");
35. fprintf(txt, "--DELETE STUDENT--\n"
36. "Student %s not found\n\n",
37. deleteName);
38. fclose(txt);
39. }
40. }
41. void searchName() {
42. int stud\_count = getStudentCount();
43. if (stud\_count == 0) {
44. cout << "Database is empty\n";
45. FILE \*txt = fopen("log.txt", "at");
46. fprintf(txt, "--LINEAR SEARCH BY NAME--\n"
47. "Database is empty\n\n");
48. fclose(txt);
49. return;
50. }

**Продолжение приложения А**

1. Student \*studs = getStudentArray(stud\_count);
2. char findName[40];
3. bool success = false;
4. cout << "Enter student name for find: ";
5. cin.getline(findName, 40);
7. FILE \*txt;
8. for(int i = 0; i < stud\_count; i++) {
9. if(nameComp(studs[i].name, findName) == 0) {
10. if(!success) {
11. txt = fopen("log.txt", "at");
12. fprintf(txt, "--LINEAR SEARCH BY NAME--\n");
13. logTable(txt);
14. printTable();
15. }
16. printStudent(&studs[i]);
17. logOutputStudent(txt, studs[i]);
18. success = true;
19. }
20. }
21. delete[] studs;
23. if(!success) {
24. cout << "No such student\n";
25. txt = fopen("log.txt", "at");
26. fprintf(txt, "--LINEAR SEARCH BY NAME--\n"
27. "Student %s not found\n\n",findName);
28. }
29. else
30. fprintf(txt, "\n");
31. cout << endl;
32. fclose(txt);
33. }
34. void logOutputStudent(FILE \*txt, Student &student) {
35. fprintf(txt, "%-40s%-10d%10.2f%15s%20d\n",
36. student.name, student.group, student.score,
37. (student.activist) ? "yes" : "no", student.income);}
38. void logTable(FILE \*txt) {
39. fprintf(txt, "%-40s%-10s%10s%15s%20s\n", " name",
40. "group", "av. score", "activist", "income");
41. }
42. void logOutputArray(FILE \*txt, Student \*studs,
43. int stud\_count) {
44. for(int i = 0; i < stud\_count; i++)
45. logOutputStudent(txt, studs[i]);
46. }
47. void sortGroup() {
48. int stud\_count = getStudentCount();
49. if (stud\_count == 0) {
50. cout << "Database is empty\n";
51. FILE \*txt = fopen("log.txt", "at");
52. fprintf(txt, "--SELECTION SORT BY GROUP--\n"
53. "Database is empty\n\n");

**Продолжение приложения А**

1. fclose(txt);
2. return;
3. }
4. Student \*studs = getStudentArray(stud\_count);
5. selectionSortByGroup(studs, stud\_count);
6. FILE \*txt = fopen("log.txt", "at");
7. fprintf(txt, "--SELECTION SORT BY GROUP--\n");
8. logTable(txt);
9. logOutputArray(txt, studs, stud\_count);
10. fprintf(txt, "\n");
11. fclose(txt);
13. printTable();
14. printStudentArray(studs, stud\_count);
15. cout << endl;
16. delete[] studs;
17. }
18. void selectionSortByGroup(Student \*studs,
19. int stud\_count) {
20. for(int i = 0; i < stud\_count - 1; i++) {
21. int minIndex = i;
22. for(int j = i+1; j < stud\_count; j++)
23. if(studs[j].group < studs[minIndex].group)
24. minIndex = j;
26. if(minIndex != i) {
27. Student temp = studs[i];
28. studs[i] = studs[minIndex];
29. studs[minIndex] = temp;
30. }
31. }
32. }
33. void searchGroup() {
34. int stud\_count = getStudentCount();
35. if (stud\_count == 0) {
36. cout << "Database is empty\n";
37. FILE \*txt = fopen("log.txt", "at");
38. fprintf(txt, "--BINARY SEARCH BY GROUP--\n"
39. "Database is empty\n\n");
40. fclose(txt);
41. return;
42. }
43. Student \*studs = getStudentArray(stud\_count);
44. selectionSortByGroup(studs, stud\_count);
45. int find\_group = inputNewGroup();
46. cin.ignore();
47. int group\_index = binarySearchByGroup(studs,
48. stud\_count, find\_group);

**Продолжение приложения А**

1. if(group\_index != -1) {
2. FILE \*txt = fopen("log.txt", "at");
3. printTable();
4. fprintf(txt, "--BINARY SEARCH BY GROUP--\n");
5. logTable(txt);
6. for(int i = group\_index;
7. studs[i].group == find\_group; i++) {
8. logOutputStudent(txt, studs[i]);
9. printStudent(&studs[i]);
10. }
11. fprintf(txt, "\n");
12. fclose(txt);
13. }
14. else {
15. FILE \*txt = fopen("log.txt", "at");
16. fprintf(txt, "--BINARY SEARCH BY GROUP--\n"
17. "Group %d students not found\n\n",
18. find\_group);
19. fclose(txt);
20. cout << "No such student\n";
21. }
22. cout << endl;
23. delete[] studs;
24. }
25. int binarySearchByGroup(Student \*studs, int stud\_count,
26. int find\_group) {
27. int start = 0;
28. int len = stud\_count;
29. while(len > 0) {
30. int half = len / 2;
31. if (studs[start + half].group < find\_group) {
32. start += half + 1;
33. len -= half + 1;
34. }
35. else
36. len = half;
37. }
38. if (start < stud\_count
39. && studs[start].group == find\_group)
40. return start;
41. else
42. return -1;
43. }
44. void sortAvScore() {
45. int stud\_count = getStudentCount();
46. if (stud\_count == 0) {
47. cout << "Database is empty\n";
48. FILE \*txt = fopen("log.txt", "at");
49. fprintf(txt, "--INSERTION SORT BY AV. SCORE--\n"
50. "Database is empty\n\n");
51. fclose(txt);
52. return;
53. }

**Продолжение приложения А**

1. Student \*studs = getStudentArray(stud\_count);
2. for(int i = 1; i < stud\_count; i++) {
3. Student key = studs[i];
4. int j;
5. for(j = i - 1; j >=0
6. && studs[j].score < key.score; j--)
7. studs[j+1] = studs[j];
8. studs[j+1] = key;
9. }
10. FILE \*txt = fopen("log.txt", "at");
11. fprintf(txt, "--INSERTION SORT BY AV. SCORE--\n");
12. logTable(txt);
13. logOutputArray(txt, studs, stud\_count);
14. fprintf(txt, "\n");
15. fclose(txt);
16. printTable();
17. printStudentArray(studs, stud\_count);
18. cout << endl;
20. delete[] studs;
21. }
22. void sortName() {
23. int stud\_count = getStudentCount();
24. if (stud\_count == 0) {
25. cout << "Database is empty\n";
26. FILE \*txt = fopen("log.txt", "at");
27. fprintf(txt, "--QUICK SORT BY NAME--\n"
28. "Database is empty\n\n");
29. fclose(txt);
30. return;
31. }
32. Student \*studs = getStudentArray(stud\_count);
33. quicksortByName(studs, 0, stud\_count-1);
34. FILE \*txt = fopen("log.txt", "at");
35. fprintf(txt, "--QUICK SORT BY NAME--\n");
36. logTable(txt);
37. logOutputArray(txt, studs, stud\_count);
38. fprintf(txt, "\n");
39. fclose(txt);
40. printTable();
41. printStudentArray(studs, stud\_count);
42. cout << endl;
43. delete[] studs;
44. }
45. void quicksortByName(Student \*studs, int l, int r) {
46. Student pivot = studs[(l+r) / 2], temp\_stud;
47. int lt = l, rt = r; // left temp & right temp
48. do {
49. while(nameComp(studs[lt].name, pivot.name) < 0)

**Продолжение приложения А**

1. lt++;
2. while(nameComp(studs[rt].name, pivot.name) > 0)
3. rt--;
4. if(lt <= rt) {
5. temp\_stud = studs[lt];
6. studs[lt] = studs[rt];
7. studs[rt] = temp\_stud;
8. lt++;
9. rt--;
10. }
11. } while(lt <= rt);
12. if (l < rt)
13. quicksortByName(studs, l, rt);
14. if (lt < r)
15. quicksortByName(studs, lt, r);
16. }
17. void specialSearch() {
18. int stud\_count = getStudentCount();
19. if (stud\_count == 0) {
20. cout << "Database is empty\n";
21. FILE \*txt = fopen("log.txt", "at");
22. fprintf(txt, "--SPECIAL SEARCH--\n"
23. "Database is empty\n\n");
24. fclose(txt);
25. return;
26. }
27. Student \*studs = getStudentArray(stud\_count);
28. selectionSortByGroup(studs, stud\_count);
29. int find\_group = inputNewGroup(); cin.ignore();
30. float find\_score = inputNewScore();
31. int group\_index = binarySearchByGroup(studs,
32. stud\_count, find\_group);
33. bool success = false;
34. if(group\_index != -1) {
35. int groupmates\_count = 0;
36. for(int i = group\_index;
37. studs[i].group == find\_group; i++)
38. groupmates\_count++;
39. quicksortByName(studs, group\_index,
40. group\_index+groupmates\_count-1);
42. FILE \*txt = fopen("log.txt", "at");
43. printTable();
44. fprintf(txt, "--SPECIAL SEARCH--\n");
45. logTable(txt);
46. for(int i = group\_index;
47. studs[i].group == find\_group; i++) {
48. if(studs[i].score > find\_score
49. && studs[i].activist) {
50. success = true;
51. logOutputStudent(txt, studs[i]);

**Продолжение приложения А**

1. printStudent(&studs[i]);
2. }
3. }
4. fprintf(txt, "\n");
5. fclose(txt);
6. }
7. if(!success) {
8. FILE \*txt = fopen("log.txt", "at");
9. fprintf(txt, "--SPECIAL SEARCH--\n"
10. "No %d group student "
11. "with %.2f av. score\n\n",
12. find\_group, find\_score);
13. fclose(txt);
14. cout << "No such student\n";
15. }
16. cout << endl;
17. delete[] studs;
18. }
19. void getStats() {
20. int stud\_count = getStudentCount();
21. if (stud\_count == 0) {
22. cout << "Database is empty\n";
23. FILE \*txt = fopen("log.txt", "at");
24. fprintf(txt, "--QUEUE TO DORM--\n"
25. "Database is empty\n\n");
26. fclose(txt);
27. return;
28. }
29. int min\_salary;
30. cout << "Enter minimal salary: ";
31. while (!(cin >> min\_salary) || min\_salary <= 0) {
32. cout << "Error! Enter positive number: ";
33. cin.clear();
34. cin.ignore(numeric\_limits<streamsize>::max(),'\n');
35. }
36. cin.ignore();
37. Student\* studs = getStudentArray(stud\_count);
38. insertionSortByPriority(studs, stud\_count,
39. min\_salary);
40. cout << "Priority list for dormitory:\n";
41. FILE \*txt = fopen("log.txt", "at");
42. printTable();
43. fprintf(txt, "--QUEUE TO DORM--\n");
44. logTable(txt);
46. int privileged\_count = 0;
47. for (int i = 0; i < stud\_count; i++) {
48. if (studs[i].income < 2 \* min\_salary) {
49. privileged\_count++;
50. }

**Продолжение приложения А**

1. printStudent(&studs[i]);
2. logOutputStudent(txt, studs[i]);
3. }
5. cout << "Number of privileged students: "
6. << privileged\_count << endl << endl;
7. fprintf(txt, "Privileged students: %d\n\n",
8. privileged\_count);
9. fclose(txt);
10. delete[] studs;
11. }
12. void insertionSortByPriority(Student\* studs, int count,
13. int min\_salary) {
14. for (int i = 1; i < count; i++) {
15. Student key = studs[i];
16. int j;
17. for (j = i - 1; j >= 0
18. && comparePriority(studs[j], key, min\_salary);
19. j--)
20. studs[j + 1] = studs[j];
21. studs[j + 1] = key;
22. }
23. }
24. bool comparePriority(const Student& a,
25. const Student& b,
26. int min\_salary){
27. bool a\_pr = (a.income < 2 \* min\_salary);
28. bool b\_pr = (b.income < 2 \* min\_salary);
29. if (a\_pr != b\_pr)
30. return !a\_pr;
32. if (a.score != b.score)
33. return a.score < b.score;
35. return !a.activist && b.activist;
36. }

**Файл “header.h”:**

1. #ifndef DORM\_H
2. #define DORM\_H
3. #include <iostream>
4. #include <fstream>
5. #include <iomanip>
6. #include <limits>
7. #include <cstdio>
8. struct Student {
9. char name[40];
10. int group;
11. float score;
12. bool activist;

**Продолжение приложения А**

1. int income;
2. };
3. // ----- MAIN FUNCTIONS ---- //
4. void createFile();
5. void addStudent();
6. void printAll();
7. void editStudent();
8. void deleteStudent();
9. void searchName();      // linear
10. void searchGroup();     // binary
11. void sortName();        // quicksort
12. void sortGroup();       // selection
13. void sortAvScore();     // insertion
14. void specialSearch();
15. void getStats();
16. // ----------------- INPUT/OUTPUT ------------------ //
17. // console
18. int inputNewGroup();
19. float inputNewScore();
20. bool inputNewActivist();
21. int inputNewIncome();
22. void inputStudentInfo(Student \*stud);
23. void printTable();
24. void printStudent(Student \*stud);
25. // bin file
26. Student \*getStudentArray(int stud\_count);
27. void writeStudentArray(Student \*studs, int stud\_count);
28. int getStudentCount();
29. // logging
30. void logTable(FILE \*txt);
31. void logOutputStudent(FILE \*txt, Student &student);
32. void logOutputArray(FILE \*txt, Student \*studs,
33. int stud\_count);
34. // ------------- HELPER FUNCTIONS ------------------ //
35. // search
36. Student \*findStudentByName(Student \*studs, char \*name,
37. int stud\_count);
38. int binarySearchByGroup(Student \*studs, int stud\_count,
39. int find\_group);
40. // sort
41. void selectionSortByGroup(Student \*studs,
42. int stud\_count);
43. void insertionSortByPriority(Student\* studs, int count,
44. int min\_salary);
45. void quicksortByName(Student \*studs, int low,
46. int high);
47. // compare
48. int nameComp(char \*first, char \*second);
49. bool comparePriority(const Student& a,
50. const Student& b, int min\_salary);
51. #endif

Приложение Б  
(обязательное)  
Блок-схема работы программы

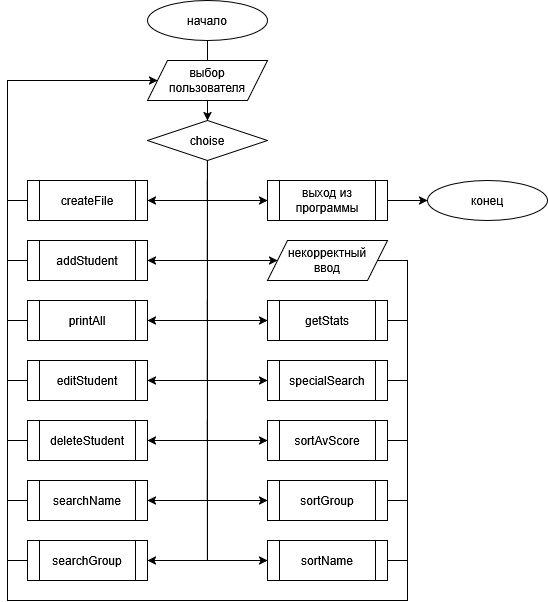


Рисунок Б.1 – Блок-схема работы программы

**ВЕДОМОСТЬ ДОКУМЕНТОВ**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнитель-ные сведения* | |
|  | | | | |  | | | |  | |
|  | | | | | *Текстовые документы* | | | |  | |
|  | | | | |  | | | |  | |
| *БГУИР КП 7-06-0611-03 053 ПЗ* | | | | | *Пояснительная записка* | | | | *45 с.* | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | ***БГУИР КП 7-06-0611-03 053 ВД*** | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л.* | *№ докум.* | *Подп.* | *Дата* | *Система учета заселения в общежитие* | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Мацукевич* |  |  |  | *У* |  | *45* | *45* |
| *Проверил* | | *Новицкая* |  |  | *Кафедра ВМиП*  *гр. 421703* | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |